Universidad Autónoma de Madrid
Escuela politécnica superior

Doble Grado en Ingeniería Informática y Matemáticas

Trabajo Fin de Grado

APRENDIZAJE MULTI-TAREA MEDIANTE PROCESOS GAUSSIANOS PARA CLASIFICACIÓN

Autor: Víctor Velasco Pardo
Tutor: Daniel Hernández Lobato

Junio 2016

APRENDIZAJE MULTI-TAREA MEDIANTE PROCESOS GAUSSIANOS PARA CLASIFICACIÓN

Autor: Víctor Velasco Pardo
Tutor: Daniel Hernández Lobato

Dpto. de Ingeniería Informática
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Junio 2016

Resumen

Resumen

El aprendizaje multi-tarea es una técina de aprendizaje automático que consiste en entrenar un algoritmo de aprendizaje automático para aprender múltiples tareas usando una representación compartida para todas ellas.

Aunque el enfoque tradicional en el campo del aprendizaje automático consiste en aprender una única tarea, en ciertos problemas aprender varias tareas relacionadas al mismo tiempo aumenta la capacidad predictiva. f En el presente trabajo de fin de grado nos proponemos comprobar la utilidad del aprendizaje multi-tarea a la hora de afrontar problemas de clasificación. Para ello utilizaremos los modelos conocidos como *procesos gaussianos*, que recientemente han tenido una gran acogida en la comunidad de investigadores en el campo del aprendizaje automático.

Para reconocer patrones, los procesos gaussianos necesitan de una función núcleo (o kernel) suficientemente expresivo. En este trabajo utilizamos un kernel conocido como *Spectral Mixture Kernels*, propuestos por A. Wilson y R. Adams. Estos kernels tienen una propiedad que los hacen especialmente útiles: podemos aproximar cualquier función de covarianza estacionaria con una precisión arbitraria.

Palabras Clave

Aprendizaje automático, clasificación, aprendizaje multi-tarea, procesos gaussianos

Abstract

Multi-task learning is a technique within the field of machine learning which uses a machine learning algorithm so as to learn multiple tasks at the same time using a shared representation for them all.

Even though the traditional approach in the machine learning field is to learn one single task a time, it turns out that in order to solve some problems learning different but related tasks at the same time boosts predictive capacity.

In this Bachelor's thesis we try to show the uselfulness of multi-task learning to tackle classification problems. To do so, we will use the models known as *gaussian processes*, which recently have won popularity within the machine learning research community.

In order to recognize patterns, gaussian processes need an expressive enough *kernel function*. In the present thesis we use a family of kernels known as *Spectral Mixture Kernels*, propesed by A. Wilson and R. Adams. These kernels are endowed with a property that makes them specially useful: we can approximate any *stationary* kernel to a SM Kernel with arbitrary precision.

Key words

Machine learning, classification, multi-task learning, gaussian processes

Índice general

Índice de Figuras

Índice de Tablas

1

Introducción y conceptos básicos

1.1. Motivación

Durante la última década, dentro del campo del *aprendizaje automático*, se ha producido una enorme cantidad de trabajo alrededor de los métodos *kernel*. Posiblemente el mejor ejemplo sea el de las *máquinas de soporte vectorial* (SVM por sus siglas en inglés). Otro de los modelos más populares dentro de los métodos kernel ha sido el de los *procesos gaussianos* (GP), que son objeto de estudio del presente trabajo [1, Capítulo 1].

Aunque el uso de los procesos gaussianos en el ámbito del aprendizaje automático es muy reciente, estos modelos se han utilizado en estadística desde los últimos años del siglo XIX, fundamentalmente en el campo de la *geoestadística*. Podemos clasificar los GPs dentro del grupo de los métodos kernel *probabilísticos*, que de momento han encontrado menos aceptación en la industria que los métodos kernel *no probabilísticos*, como SVM. Esto puede ser debido a una de las principales desventajas de los GPs: el alto coste computational de utilizarlos con conjuntos de datos de gran tamaño.

El interés en los procesos gaussianos por parte de la comunidad investigadora en aprendizaje automático surge a partir de la madurez de las *redes neuronales*, ampliamente aceptadas en este campo. El número de funciones que puede aprender una red neuronal de dos capas está limitado por el número de *unidades ocultas* de la red neuronal. Aumentando el número de unidades ocultas podemos aumentar el número de funciones que una red neuronal puede aprender. Sin embargo, esto causaría un problema de *overfitting* o *sobreajuste*, que limita el número de capas ocultas que una red neuronal puede tener. Aquí aparece el interés de los GPs: se ha demostrado que la distribución de funciones generada por una red neuronal tiende a un GP cuando el número de capas ocultas tiende a infinito.

Otro campo de investigación que ha generado interés en los últimos años dentro del aprendizaje automático es el del *aprendizaje multi-tarea* (MTL, por sus siglas en inglés). El aprendizaje multi-tarea es un enfoque que pretende mejorar el rendimiento (entendido como una mejora en la *tasa de acierto*) de los algoritmos de aprendizaje supervisado aprendiendo en paralelo diferentes tareas *relacionadas*, de forma que puedan utilizar una representación del conocimiento aprendido: lo que se aprende para una tarea puede ayudar a aprender mejor otras tareas. Se ha comprobado, por ejemplo en [2] que esta idea funciona.

Dentro del campo del aprendizaje supervisado, encontramos dos tipos de problemas: los

problemas de *regresión*, en los que el objetivo es aprender una función continua, y los problemas de *clasificación*, en los que se pretende categorizar observaciones en una *clase* o *grupo* dentro de dos o más posibles clases. En este trabajo nos centraremos en estos últimos, con el objetivo de comprobar si el enfoque de MTL ayuda o no a mejorar la capacidad predictiva de los GPs.

Por último, cabe mencionar una propiedad importante de los métodos kernel. Dentro del aprendizaje supervisado, encontramos fundamentalmente dos tipos de métodos: *paramétricos* y *no paramétricos*. Los primeros asumen *a priori* una distribución para los datos, que es función de unos *parámetros* que se determinan a partir de los datos. El ejemplo canónico es el del método de *regresión lineal*: queremos aprender una variable respuesta $y(x)$ a partir de una variable regresora x, y asumimos que la variable respuesta depende linealmente de la regresora, es decir, que $y(x) = ax + b$, siendo a y b parámetros que debemos estimar a partir de los datos. Los GPs y, más generalemente, los métodos kernel, son métodos *no paramétricos* ya que no asumen que la distribución a la que se ajustan los datos tienen un número fijo de parámetros. Sin embargo, es necesario elegir un *kernel* o función de covarianza que determinará la solución al problema. En este sentido, estudiaremos unos kernels con la propiedad de ser *muy expresivos*, es decir, que pueden describir patrones muy diferentes en los datos ya que pueden aproximar arbitrariamente kernels muy diferentes [3].

1.2. Objetivos y enfoque

El primer objetivo de este trabajo es hacer un estudio teórico de los procesos gaussianos, de su uso para problemas de clasificación y del aprendizaje multi-tarea, así como de la posibilidad de utilizar GPs para aprender varias tareas en paralelo.

El segundo objetivo es realizar diferentes experimentos que nos permitan comprobar la utilidad del aprendizaje multi-tarea cuando se utiliza un modelo de GPs. El principal problema que nos encontraremos entonces será el tiempo de ejecución que requiere el aprendizaje con GPs: para aprender los hiperparámetros necesitaremos utilizar un método para aproximar integrales que no se pueden calcular analíticamente. Los métodos disponibles son *aproximación de Laplace*, *expectation propagation* y *aproximaciones de Monte Carlo con cadenas de Markov*. Estos métodos tienen ventajas e inconvenientes, pero todos ellos tienen un tiempo de ejecución $\mathcal{O}(n^3)$, siendo n el número de datos de entrenamiento, lo que nos obligará a trabajar con conjuntos de datos pequeños o bien a utilizar alguno de los métodos de *inferencia aproximada* citados anteriormente.

La implementación de los algoritmos y el tratamiento de los datos necesario para realizar los experimentos se realizará utilizando el lenguaje de programación **Python**, que es uno de los lenguajes más utilizados en el campo del aprendizaje automático y es de gran utilidad para el análisis de datos. El lenguaje Python, siendo un lenguaje de propósito general, ofrece una sintaxis muy sencilla, próxima al pseudocódigo y a la de otros lenguajes orientados al cálculo científico y el análisis de datos, como Matlab o R.

1.3. Conceptos básicos

1.3.1. Aprendizaje supervisado

El *aprendizaje automático* es un área de la informática que pretende resolver el problema de *identificar patrones en un conjunto de datos*. Dentro del campo del aprendizaje automático, encontramos tres enfoques principales: *aprendizaje supervisado*, *aprendizaje no supervisado* y *aprendizaje por refuerzo* [4, Capítulo 1].

El *aprendizaje supervisado* es el que estudiaremos en este trabajo y sus características las detallaremos más adelante. El objetivo del *aprendizaje no supervisado* es descubrir grupos de ejemplos en un conjunto de datos, determinar la distribución de un conjunto de datos dentro del dominio de los *inputs* o proyectar un conjunto de datos en un espacio de dimensión grande a uno de dimensión dos o tres con el objetivo de facilitar su *visualización*. Por último, el objetivo del *aprendizaje por refuerzo* es optimizar las acciones que se deben tomar en una situación determinada con el objetivo de maximizar su *utilidad*. En este último caso el algoritmo de aprendizaje no recibe ejemplos de entrenamiento, sino que debe descubrir las acciones óptimas mediante una técnica de *ensayo y error*.

Dados un *conjunto de entrenamiento* $\{\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_N\}$ y un *vector de observaciones* $\boldsymbol{y} = (y_1, y_2, ..., y_N)^T$, el objetivo del *aprendizaje supervisado* es aprender una función f. Si el objetivo es aprender una función continua, estamos ante un problema de *regresión*. Si, por el contrario, la función toma valores en un conjunto finito, estamos ante un problema de *clasificación*. En este trabajo nos centraremos en estos últimos.

En un problema de regresión, asumimos que el *vector de observaciones* $\boldsymbol{y}$ es el resultado de evaluar una función f en los vectores $\boldsymbol{x}_i$ y añadir un cierto *ruido*. La función f es desconocida y el objetivo es encontrar el valor de f en un dato *de prueba* $\boldsymbol{x}_*$ cometiendo un error tan pequeño como sea posible. Normalmente asumiremos que la distribución del ruido es normal, es decir, asumimos que:

$$y(\boldsymbol{x}_i) = f(\boldsymbol{x}_i) + \epsilon_i, \text{ con } \epsilon_i \sim \mathcal{N}(0, \sigma^2) \tag{1.1}$$

En un problema de clasificación, asumimos que el *vector de observaciones* $\boldsymbol{y}$ es el resultado de evaluar una *función latente* f en los vectores $\boldsymbol{x}_i$, añadir un cierto *ruido* ϵ y por último aplicar una regla y como sigue:

$$y(f(\boldsymbol{x}_i) + \epsilon) = \begin{cases} 1 & f(\boldsymbol{x}_i) + \epsilon \geq 0 \\ -1 & f(\boldsymbol{x}_i) + \epsilon < 0 \end{cases} \tag{1.2}$$

Normalmente consideramos que el ruido ϵ es logístico, es decir, $\epsilon \sim Logistic(0, s)$.

Siguiendo [1], representamos nuestro conjunto de entrenamiento, $\mathcal{D} = \{(\boldsymbol{x}_i, y_i) | i = 1, 2, ..., n\}$, donde n es el número de observaciones (todas ellas con dimensión D), con una *matriz de diseño* de tamaño $D \times n$, X, en la que cada columna es un vector x_i y un vector columna $\boldsymbol{y}$, en el que cada elemento es la observación correspondiente al vector $\boldsymbol{x}_i$. De esta forma, podemos escribir $\mathcal{D} = (X, \boldsymbol{y})$ La matriz X se conoce en la literatura académica como *matriz de diseño* y es más habitual en la literatura utilizar la traspuesta de $\mathcal{D}$. Es decir, es más habitual utilizar una matriz cuyas filas son los vectores $\boldsymbol{x}_1, ..., \boldsymbol{x}_n$. El enfoque que utilizamos aquí tiene la ventaja de que cada dato es un vector columna.

1.3.2. Conceptos matemáticos

Antes de comenzar, recordamos conceptos básicos de probabilidad y estadística, álgebra lineal y optimización, siguiendo [4, Capítulo 2] y [1, Apéndice A].

Conceptos básicos de probabilidad y estadística

Definición 1.1. Decimos que el vector aleatorio continuo $\boldsymbol{x}$ sigue una **distribucion normal multivariante D-dimensional** con vector media $\boldsymbol{\mu}$ y matriz de covarianzas $\boldsymbol{\Sigma}$ si su función de densidad evaluada en un vector de dimensión D $\boldsymbol{x}$ es:

$$\mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{D/2}} \frac{1}{|\Sigma|^{1/2}} \exp(\frac{1}{2}(\boldsymbol{x} - \boldsymbol{\mu})^T \Sigma^{-1} (\boldsymbol{x} - \boldsymbol{\mu})) \tag{1.3}$$

La distribución normal multivariante es importante porque aparece de manera natural en muchas ocasiones. En particular, el *teorema central del límite* nos dice que, dadas ciertas condiciones, la suma de N variables aleatorias se aproxima a una distribución normal para N suficientemente grande.

En ocasiones tenemos dos vectores aleatorios $\boldsymbol{x}_a, \boldsymbol{x}_b$ cuya distribución *conjunta* es normal y estamos interesados en la distribución de $\boldsymbol{x}_a$ *condicionada* a $\boldsymbol{x}_b$. Por ejemplo, supongamos que tenemos un vector de observaciones $\boldsymbol{f}$ y un dato de prueba $\boldsymbol{x}_*$. Estamos interesados en predecir $f_* = f(\boldsymbol{x}_*)$. Para ello, estamos interesados en conocer con que probabilidad toma f_* un valor determinado, dadas las observaciones $\boldsymbol{f}$. Es decir, estamos interesados en conocer $p(f_*|\boldsymbol{f})$.

Supongamos que tenemos una distribución normal conjunta $\mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma})$ y definimos una partición del vector $\boldsymbol{x}$ de la siguiente manera:

$$\boldsymbol{x} = \begin{pmatrix} \boldsymbol{x}_a \\ \boldsymbol{x}_b \end{pmatrix}, \boldsymbol{\mu} = \begin{pmatrix} \boldsymbol{\mu}_a \\ \boldsymbol{\mu}_b \end{pmatrix}, \boldsymbol{\Sigma} = \begin{pmatrix} \boldsymbol{\Sigma}_{aa} & \boldsymbol{\Sigma}_{ab} \\ \boldsymbol{\Sigma}_{ba} & \boldsymbol{\Sigma}_{bb} \end{pmatrix}, \tag{1.4}$$

De esta forma, podemos calcular la distribución de $\boldsymbol{x}_a$ condicionada a $\boldsymbol{x}_b$:

$$p(\boldsymbol{x}_a|\boldsymbol{x}_b) = \mathcal{N}(\boldsymbol{x}_a|\boldsymbol{\mu}_{a|b}, \boldsymbol{\Sigma}_{a|b}) \tag{1.5}$$

$$\boldsymbol{\mu}_{a|b} = \boldsymbol{\mu}_a + \boldsymbol{\Sigma}_{ab}\boldsymbol{\Sigma}_{bb}^{-1}(\boldsymbol{x}_b - \boldsymbol{\mu}_b) \tag{1.6}$$

$$\boldsymbol{\Sigma}_{a|b} = \boldsymbol{\Sigma}_{aa} - \boldsymbol{\Sigma}_{ab}\boldsymbol{\Sigma}_{bb}^{-1}\boldsymbol{\Sigma}_{ba} \tag{1.7}$$

En ocasiones conocemos la distribución *conjunta* de un vector $(\boldsymbol{x}_a, \boldsymbol{x}_b)^T$ distribuido normalmente y estamos interesados en conocer la distribución *marginal* del vector $\boldsymbol{x}_a$. En ese caso:

$$p(\boldsymbol{x}_a) = \mathcal{N}(\boldsymbol{x}_a|\boldsymbol{\mu}_a, \boldsymbol{\Sigma}_{aa}) \tag{1.8}$$

Un resultado agradable y que nos ayudará más adelante a calcular la *verosimilitud marginal* de un proceso gaussiano es que el producto de dos distribuciones normales es una distribución normal *no normalizada*, es decir, es una distribución normal multiplicada por una *constante de normalización* Z^{-1}:

$$\mathcal{N}(\boldsymbol{x}|\boldsymbol{a}, A)\mathcal{N}(\boldsymbol{x}|\boldsymbol{b}, B) = Z^{-1}\mathcal{N}(\boldsymbol{x}|\boldsymbol{c}, C) \tag{1.9}$$

$$Z^{-1} = (2\pi)^{-D/2}|A + B|^{-1/2}\exp(-\frac{1}{2}(\boldsymbol{a} - \boldsymbol{b})^T(A + B)^{-1}(\boldsymbol{a} - \boldsymbol{b})) \tag{1.10}$$

Utilizando el resultado anterior y sabiendo que la integral de una función de densidad es uno, obtenemos el siguiente resultado, que nos ayudará a obtener una expresión cerrada para la *verosimilitud marginal*:

$$\int \mathcal{N}(\boldsymbol{x}|\boldsymbol{a}, A)\mathcal{N}(\boldsymbol{x}|\boldsymbol{b}, B)d\boldsymbol{x} = Z^{-1}\int \mathcal{N}(\boldsymbol{x}|\boldsymbol{c}, C)d\boldsymbol{x} = Z^{-1} \tag{1.11}$$

La distribución normal tiene algunas desventajas que producen limitaciones a la hora de modelizar algunos conjuntos de datos. Con el objetivo de superar estas limitaciones, podemos trabajar con combinaciones lineales de distribuciones normales, llamadas *mixturas*. Definimos una mixtura de K distribuciones normales como la distribución de probabilidad que tiene la siguiente función de masa:

$$p(\boldsymbol{x}) = \sum_{k=1}^{K} w_k\mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k), \tag{1.12}$$

siendo $\mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$ la función de masa de una distribución normal con media $\boldsymbol{\mu}_k$ y covarianza $\boldsymbol{\Sigma}_k$. Cada una de estas K distribuciones es una *componente* de la mixtura. Ya que $p(\boldsymbol{x})$ debe verificar las propiedades de las distribuciones de probabilidad, es fácil verificar que:

$$0 \leq w_k \leq 1 \qquad \sum_{k=1}^{K} w_k = 1 \tag{1.13}$$

En la siguiente imagen se ilustra el concepto de mixtura de gaussianas: en azul y verde se muestran dos distribuciones gaussianas, y en rojo una mixtura de ambas.

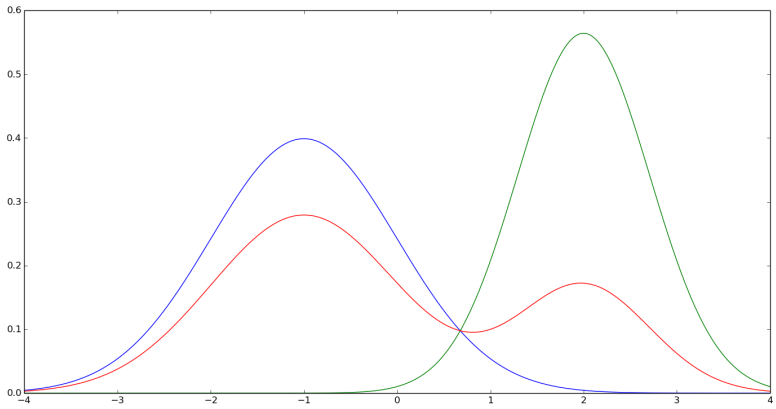


Figura 1.1: Ilustración del concepto de mixtura de gaussianas.

Conceptos básicos de álgebra lineal

El siguiente lema de álgebra lineal nos ayudará a implementar algunos algoritmos de forma eficiente:

Lema 1.2. *Sean Z, W, U y V matrices de tamaño $n \times n$, $m \times m$, $n \times m$ y $n \times m$ respectivamente. Supongamos además que las inversas Z^{-1}, W^{-1}, U^{-1} y V^{-1} existen. Entonces se verifica la siguiente igualdad:*

$$(Z + UWV^T)^{-1} = Z^{-1} - Z^{-1}U(W^{-1} + V^T Z^{-1} U)^{-1} V^T Z^{-1} \tag{1.14}$$

Este lema se conoce habitualmente como *lema de inversión de matrices* o *fórmula de Woodbury-Sherman-Morrison*. A nosotros nos vale con una versión más sencilla:

Lema 1.3. *Sean Z y W dos matrices invertibles de tamaño $n \times n$. Entonces se verifica la siguiente igualdad:*

$$(Z + W)^{-1} = Z^{-1} - Z^{-1}(W^{-1} + Z^{-1})^{-1} Z^{-1} \tag{1.15}$$

Un resultado similar nos ayudará a calcular algunos determinantes:

Lema 1.4. *Sean Z, W, U y V matrices de tamaño $n \times n$, $m \times m$, $n \times m$ y $n \times m$ respectivamente. Supongamos además que las inversas Z^{-1}, W^{-1}, U^{-1} y V^{-1} existen. Entonces se verifica la siguiente igualdad:*

$$|Z + UWV^T| = |Z||W||W^{-1} + V^T Z^{-1} U| \tag{1.16}$$

Más adelante también necesitaremos calcular la derivada de una matriz K respecto de una variable real θ. Esta derivada se define como la matriz de derivadas de cada elemento de K, y la representamos como $\frac{\partial K}{\partial \theta}$.

A la hora de calcular derivadas de matrices, hay dos identidades que nos resultarán útiles:

$$\frac{\partial}{\partial \theta} K^{-1} = -K^{-1} \frac{\partial K}{\partial \theta} K^{-1} \tag{1.17}$$

$$\frac{\partial}{\partial \theta} \log |K| = tr(-K^{-1} \frac{\partial K}{\partial \theta}) \tag{1.18}$$

Conceptos básicos de optimización

Al contrario de lo que sucede con el algoritmo de GPs para regresión, y con otros más sencillos como los de regresión lineal o polinómica, no es posible encontrar una fórmula analítica cerrada para la solución a un problema de clasificación con GPs. Más adelante tendremos que utilizar el un método de optimización para encontrar el máximo de una función. Con ese objetivo pasamos a describir el **método de Newton-Raphson**.

Con el objetivo de encontrar el valor de $\boldsymbol{w}$ que minimiza el valor de una función $E(\boldsymbol{w})$, inicializamos el valor de $\boldsymbol{w}$ a un valor cualquiera $\boldsymbol{w}^{(0)}$ y aplicamos la siguiente iteración:

$$\boldsymbol{w}^{(k+1)} \leftarrow \boldsymbol{w}^{(k)} - \boldsymbol{H}^{-1} \nabla E(\boldsymbol{w}) \tag{1.19}$$

Siendo $\nabla E(\boldsymbol{w})$ el gradiente de la función E (vector de las derivadas parciales) y $\boldsymbol{H} = \nabla \nabla E(\boldsymbol{w})$ el hessiano de E (matriz de las derivadas parciales segundas).

El método de Newton-Raphson es efectivo y garantiza la convergencia a un valor de $\boldsymbol{w}$ en el que E alcanza un mínimo local si el la función es dos veces diferenciable.

El método de Newton-Raphson presenta, no obstante, algunos inconvenientes. El principal de ellos es que el cálculo de la matriz hessiana de derivadas segundas $\boldsymbol{H}$ es costoso y más aún el cálculo de su inversa.

En los **métodos de cuasi-Newton** la matriz hessiana no tiene que ser calculada directamente. En su lugar se utiliza una aproximación. Uno de los métodos más importantes, que utilizaremos en este trabajo, es el método **L-BFGS** o **LM-BFGS**. Es un algoritmo que pertenece a la familia de los **Broyden-Fletcher-Goldfarb-Shanno** y cuya principal ventaja es que utiliza una cantidad de memoria muy limitada. Su uso es muy habitual en el campo del aprendizaje automático para optimizar parámetros.

1.4. Metodología y plan de trabajo

En el capítulo 2 explicaremos, desde un punto de vista teórico, los procesos gaussianos, su aplicación a problemas de clasificación y el aprendizaje multi-tarea, así como otros conceptos que serán necesarios como las funciones de covarianza (en particular los *Spectral Mixture Kernels*) o la aproximación de Laplace.

En el capitulo 3 discutimos la metodología del aprendizaje multi-tarea y explicamos como puede utilizarse en el contexto de los procesos gaussianos.

En el capítulo 4 se detallan los algoritmos que se han implementado utilizando el lenguaje Python. Se detallarán las técnicas utilizadas para conseguir un rendimiento óptimo, así como para evitar inestabilidades en la computación.

En el capítulo 5 pasaremos de la teoría a la práctica y se detallaran los experimentos realizados con el objetivo de verificar experimentalmente la utilidad de la metodología y los modelos presentados en los capítulos 2 y 3.

Por último, en el capítulo 6 se detallarán las conclusiones, así como propuestas para un posible trabajo futuro.

2

Procesos gaussianos

2.1. Introducción

El objetivo de esta sección es definir el concepto de *proceso gaussiano*, que puede entenderse como una generalización de una *distribución normal multivariante* a una distribción normal sobre un *espacio de funciones* con *dimensión infinita*. Un proceso gaussiano es un *proceso estocástico*, es decir, es una distribución sobre un espacio de funciones $\{f(\boldsymbol{x})\}$ tal que para un conjunto arbitrario de puntos $\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_n$ los valores de $f(\boldsymbol{x}_1), f(\boldsymbol{x}_2), ..., f(\boldsymbol{x}_n)$ siguen una distribución que está definida de manera consistente. Pasamos a definir más formalmente el concepto de proceso gaussiano, siguiendo [1, Capítulo 2]

Definición 2.1. Un proceso gaussiano es una colección de variables aleatorias tal que cualquier subconjunto finito de ellas tienen una distribución normal conjunta.

Un proceso gaussiano está definido por una *función media* $m(\boldsymbol{x})$ y una *función de covarianza* $k(\boldsymbol{x}, \boldsymbol{x'})$. De esta forma, podemos escribir:

$$f(\boldsymbol{x}) \sim \mathcal{GP}(m(\boldsymbol{x}), k(\boldsymbol{x}, \boldsymbol{x'})) \tag{2.1}$$

Si los vectores de entrada $\boldsymbol{x}_i$ tienen dimensión dos, entonces se dice que $f(\boldsymbol{x})$ es un *campo aleatorio gaussiano*.

2.2. Funciones kernel

Definición 2.2. Sea $\phi(\cdot)$ una aplicación $\mathbb{R}^D \longrightarrow \mathbb{R}^M$, definimos una **función kernel** $k(\cdot, \cdot) : \mathbb{R}^D \times \mathbb{R}^D \longrightarrow \mathbb{R}$ de la siguiente manera:

$$k(\boldsymbol{x}, \boldsymbol{x'}) = \phi(\boldsymbol{x})^T \phi(\boldsymbol{x'}) \tag{2.2}$$

El concepto de función kernel nos ayudará a definir un proceso gaussiano, y en el contexto de estos modelos el concepto de función kernel es equivalente al de *función de covarianza*. Un

tipo de kernels especialmente importantes para este trabajo son aquellos que son invariantes a traslaciones, conocidos como estacionarios:

Definición 2.3. Sea $k(\cdot,\cdot)$ una función kernel. Decimos que k es **estacionaria** si es función de la diferencia entre sus dos argumentos, es decir si $k(\boldsymbol{x},\boldsymbol{x}') = k(\boldsymbol{x}-\boldsymbol{x}')$.

Las funciones de covarianza estacionarias son útiles porque son invariantes respecto de traslaciones. Si además de esto necesitamos funciones de covarianza que sean invariantes ante cualquier movimiento rígido, necesitamos funciones de covarianza isotrópicas:

Definición 2.4. Sea $k(\cdot,\cdot)$ una función kernel. Decimos que k es **isotrópica** si es función de la distancia entre sus dos argumentos, es decir si $k(\boldsymbol{x},\boldsymbol{x}') = k(||\boldsymbol{x}-\boldsymbol{x}'||)$.

Las funciones kernel representan una suposición básica en los métodos kernel, incluyendo los procesos gaussianos: dados dos datos $\boldsymbol{x}_1$ y $\boldsymbol{x}_2$, esperamos que la función f que aprenda nuestro algoritmo tome valores similares para dichos datos, es decir que $f_1 = f(\boldsymbol{x}_1) \approx f(\boldsymbol{x}_2) = f_2$.

La función kernel más sencilla que podemos construir se construye a partir de la identidad $\phi(\boldsymbol{x}) = \boldsymbol{x}$. En este caso el kernel es $k(\boldsymbol{x},\boldsymbol{x}') = \boldsymbol{x}^T\boldsymbol{x}'$ y por tanto el producto escalar canónico es una función kernel válida.

Una idea interesante es el truco conocido habitualmente como *kernel trick*: si tenemos un algoritmo en el que los datos de entrada lo hacen en forma de *producto escalar*, podemos reemplazar el producto escalar por un kernel cualquiera.

Uno de los kernels más utilizados como función de covarianza de un proceso gaussiano es el **Squared Exponential (SE) Kernel**, que tiene la siguiente forma:

$$k_{SE}(\boldsymbol{x}_p,\boldsymbol{x}_q) = \sigma_f^2 \exp(-\frac{||\boldsymbol{x}_p-\boldsymbol{x}_q||^2}{2l^2}) \tag{2.3}$$

A partir de la definición que hemos dado de función kernel, podemos deducir una propiedad muy sencilla pero muy importante: cualquier función kernel k es simétrica (es decir, $k(\boldsymbol{x},\boldsymbol{x}') = k(\boldsymbol{x}',\boldsymbol{x})$). Esta propiedad nos puede ayudar a ganar intuición sobre lo que es una función kernel: $k(\boldsymbol{x},\boldsymbol{x}')$ es una *medida* de la similitud entre los vectores $\boldsymbol{x}$ y $\boldsymbol{x}'$.

Definición 2.5. Sean n vectores $\boldsymbol{x}_1,\boldsymbol{x}_2,...,\boldsymbol{x}_n$ y una función kernel $k(\boldsymbol{.},\boldsymbol{.})$. Definimos la **matriz de Gram** K como la matriz cuadrada de dimensión $n\times n$ cuyos elementos son $K_{ij} = K(\boldsymbol{x}_i,\boldsymbol{x}_j)$.

En el contexto de los procesos gaussianos, la matriz de Gram también recibe el nombre de *matriz de covarianza*.

Recordando que una función kernel es simétrica, es fácil darse cuenta de que la matriz K es también simétrica ($K = K^T$). Además, recordando que $K_{ij} = k(\boldsymbol{x}_i,\boldsymbol{x}_j) = \phi(\boldsymbol{x}_i)^T\phi(\boldsymbol{x}_j)$, definimos la matriz Φ como una matriz de tamaño $D\times n$ tal que la i-ésima columna es $\phi(\boldsymbol{x}_i)$. Entonces podemos escribir K como el producto de matrices $\Phi^T\Phi$ y por tanto para cualquier vector $\boldsymbol{z}$ podemos escribir:

$$\boldsymbol{z}^TK\boldsymbol{z} = \boldsymbol{z}^T\Phi^T\Phi\boldsymbol{z} = (\Phi\boldsymbol{z})^T(\Phi\boldsymbol{z}) \geq 0 \tag{2.4}$$

Es decir, la matriz de covarianza K es semidefinida positiva. Normalmente necesitaremos trabajar con matrices de Gram definidas positivas, es decir, que verifiquen $\boldsymbol{z}^TK\boldsymbol{z} > 0$ para cualquier vector $\boldsymbol{z} \neq 0$. Añadiendo una cantidad pequeña ϵ a los elementos de la diagonal, podemos conseguir tal matriz:

$$\boldsymbol{z}^T(K+\epsilon I)\boldsymbol{z} = \boldsymbol{z}^TK\boldsymbol{z} + \epsilon^2||\boldsymbol{z}||^2 \geq \epsilon^2||\boldsymbol{z}||^2 > 0 \tag{2.5}$$

2.3. Hiperparámetros de una función de covarianza

Normalmente las funciones de covarianza se especifican con varios hiperparámetros *libres* que pueden tomar diferentes valores. Pasamos a estudiar el papel de los hiperparámetros y para ello utilizamos como ejemplo la función de covarianza conocida como *Squared Exponential Kernel*, que viene dada por la siguiente función:

$$k_{SE}(x_p, x_q) = \sigma_f^2 \exp(-\frac{1}{2l^2}(x_p - x_q)^2) + \sigma_n^2 \delta_{pq} \tag{2.6}$$

Donde δ_{pq} es la *función delta de Kronecker* y $\{\sigma_f^2, l, \sigma_n^2\}$ son los hiperparámetros. Estos tres hiperparámetros reciben el nombre de *varianza de la señal*, *escala de longitud* y *varianza del ruido*, respectivamente.

Para entender lo que sucede al variar los hiperparámetros del *Squared Exponential Kernel* generamos varias muestras de un proceso gaussiano $f(x) \sim \mathcal{GP}(0, k_{SE}(x, x'))$ para varios conjuntos de hiperparámetros diferentes.

Recordamos entonces la propiedad fundamental de los procesos gaussianos: dados $x_1, ..., x_N$ entonces el vector $\boldsymbol{f} = (f(x_1), ..., f(x_N))^T$ sigue una distribución normal $\mathcal{N}_N(0, K)$, siendo K la matriz de Gram definida anteriormente. Para generar una muestra por tanto elegimos mil puntos $x_1, ..., x_{1000}$ en el intervalo real $(-6, 6)$ y calculamos la matriz de Gram K para los 1000 puntos. Entonces tomamos una cinco muetras de la distribución $\boldsymbol{f} \sim \mathcal{N}_{1000}(0, K)$, obteniendo los resultados siguientes:

Empezamos tomando cinco muestras de un proceso gaussino con función de media 0 y función de covarianza $k_{SE}(x, x')$, con $(l, \sigma_f, \sigma_n) = (1{,}0, 1{,}0, 0{,}001)$:

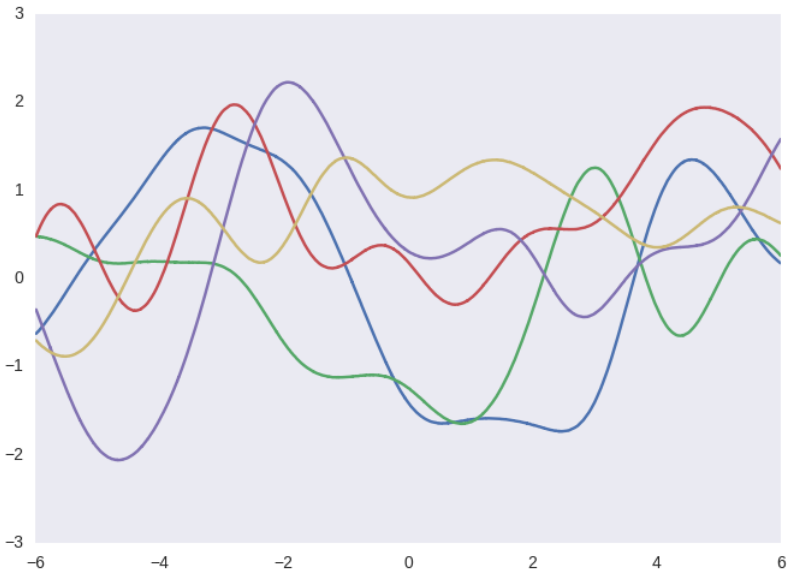


Figura 2.1: Ejemplo SE Kernel 1 $(l, \sigma_f, \sigma_n) = (1{,}0, 1{,}0, 0{,}001)$

La escala de longitud l explica la *amplitud* de la señal y por tanto las muestras serán más suaves si hacemos más grande el hiperparámetro l. Tomamos ahora cinco muestras del mismo proceso gaussiano pero esta vez con $(l, \sigma_f, \sigma_n) = (3{,}0, 1{,}0, 0{,}001)$, que se muestran en la siguiente imagen:

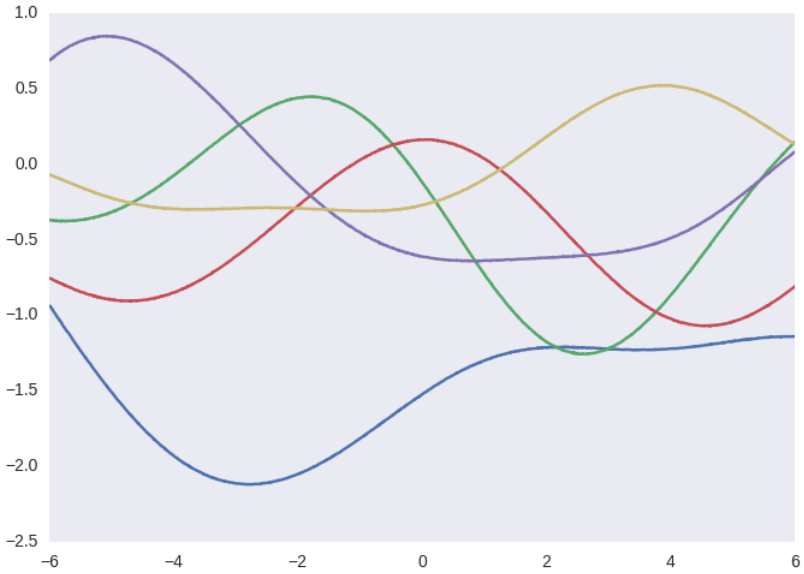


Figura 2.2: Ejemplo SE Kernel 2 $(l, \sigma_f, \sigma_n) = (3{,}0, 1{,}0, 0{,}001)$

Efectivamente las cinco funciones de nuestra muestra son ahora más suaves. Por otro lado, la *varianza de la señal* σ_f^2 explica la amplitud de la señal y al hacer más grande este hiperparámetro obtendremos como muestras funciones con una mayor amplitud:

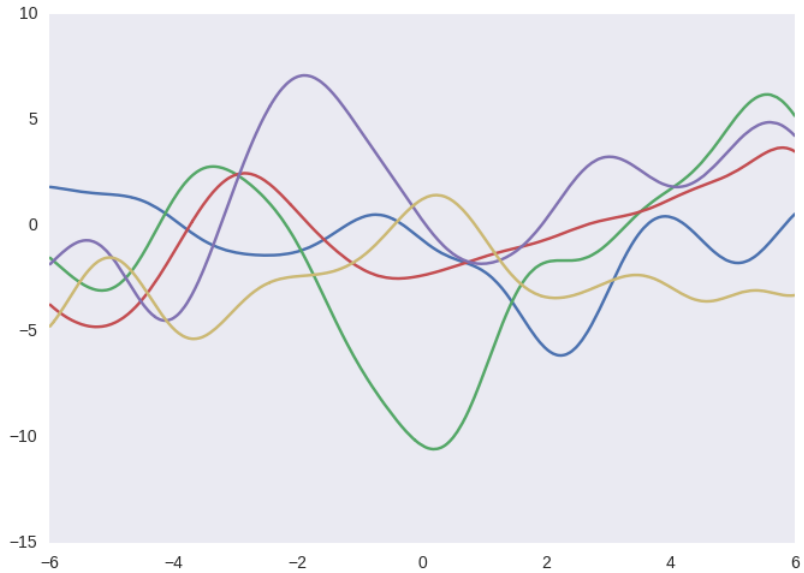


Figura 2.3: Ejemplo SE Kernel 3 $(l, \sigma_f, \sigma_n) = (1{,}0, 3{,}0, 0{,}001)$

Para la última muestra fijamos $(l, \sigma_f, \sigma_n) = (3{,}0, 0{,}25, 0{,}001)$ y obtenemos por tanto funciones más suaves que en la primera muestra y a la vez con una amplitud menor:

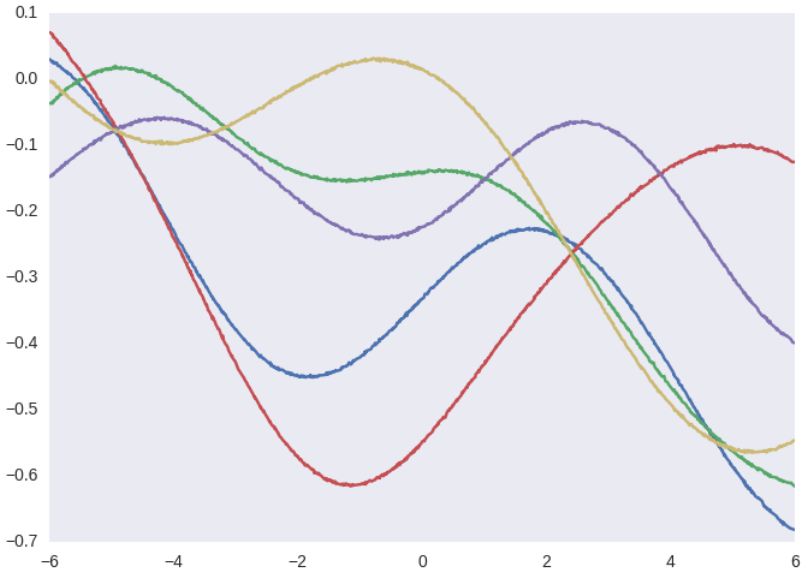


Figura 2.4: Ejemplo SE Kernel 4 $(l, \sigma_f, \sigma_n) = (3{,}0, 0{,}25, 0{,}001)$

Por supuesto, a la hora de trabajar con datos reales tomar muestras de la distribución *a priori* no nos es de mucha utilidad: además de vectores de datos $\boldsymbol{x}_1, ..., \boldsymbol{x}_N$ tendremos un vector de observaciones $\boldsymbol{y} = (y_1, ..., y_N)^T$. que nos permitirá incorporar el conocimiento que nos aportan los datos y optimizar los hiperparámetros de forma que las muestras de la *distribución posterior* se ajusten a los datos lo mejor posible.

2.4. Regresion mediante GPs

Sea $\{(\boldsymbol{x}_i, f_i)|i = 1, ..., n\}$ un conjunto de entrenamiento. Asumimos que $f_i = f(\boldsymbol{x}_i) = y(\boldsymbol{x}_i) + \epsilon_i$, con $\epsilon_i \sim \mathcal{N}(0, \sigma^2)$, y para un *dato de test* $\boldsymbol{x}_*$, queremos predecir $y(\boldsymbol{x}_*)$. A continuación buscamos la distribución conjunta de $(\boldsymbol{f}, f_*)^T$:

Sea k una función kernel definida como en 2.2. Dado un dato $\boldsymbol{x}_*$, definimos el vector $\boldsymbol{k}(\boldsymbol{x}_*)$ de la siguiente manera:

$$\boldsymbol{k}(\boldsymbol{x}_*) = (k(\boldsymbol{x}_1, \boldsymbol{x}_*), k(\boldsymbol{x}_2, \boldsymbol{x}_*), ..., k(\boldsymbol{x}_n, \boldsymbol{x}_*))^T \tag{2.7}$$

Además, la matriz de covarianza, K, para nuestro *conjunto de entrenamiento* viene dada por:

$$K = \begin{pmatrix} k(\boldsymbol{x}_1, \boldsymbol{x}_1) & k(\boldsymbol{x}_1, \boldsymbol{x}_2) & ... & k(\boldsymbol{x}_1, \boldsymbol{x}_n) \\ k(\boldsymbol{x}_2, \boldsymbol{x}_1) & k(\boldsymbol{x}_2, \boldsymbol{x}_2) & ... & k(\boldsymbol{x}_2, \boldsymbol{x}_n) \\ ... & ... & ... & ... \\ k(\boldsymbol{x}_n, \boldsymbol{x}_1) & k(\boldsymbol{x}_n, \boldsymbol{x}_2) & ... & k(\boldsymbol{x}_n, \boldsymbol{x}_n) \end{pmatrix} \tag{2.8}$$

Ahora ya podemos dar una expresión analítica para la distribución conjunta de $(\boldsymbol{f}, f_*)^T$:

$$\begin{pmatrix} \boldsymbol{f} \\ f_* \end{pmatrix} \sim \mathcal{N}\left(\boldsymbol{0}, \begin{pmatrix} K + \sigma^2 I & \boldsymbol{k}(\boldsymbol{x}_*) \\ \boldsymbol{k}(\boldsymbol{x}_*)^T & k(\boldsymbol{x}_*, \boldsymbol{x}_*) \end{pmatrix}\right) \tag{2.9}$$

Una vez que tenemos la distribución conjunta de $(\boldsymbol{f}, f_*)^T$, utiliando las fórmulas (1.5)-(1.7) obtenemos fácilmente una expresión analítica para la distribución *condicionada* de $f_*|\boldsymbol{f}$:

$$f_*|X_*, X, \boldsymbol{f} \sim \mathcal{N}(\bar{f}_*, \mathcal{V}[f_*]) \tag{2.10}$$

$$\bar{f}_* = \boldsymbol{k}_*(K + \sigma_n^2 I)^{-1}\boldsymbol{y} \tag{2.11}$$

$$\mathcal{V}[f_*] = k(\boldsymbol{x}_*, \boldsymbol{x}_*) - \boldsymbol{k}_*^T(K + \sigma_n^2 I)^{-1}\boldsymbol{k}_* \tag{2.12}$$

Para ilustrar las fórmulas 2.11 y 2.12, utilizamos las siguientes figuras:

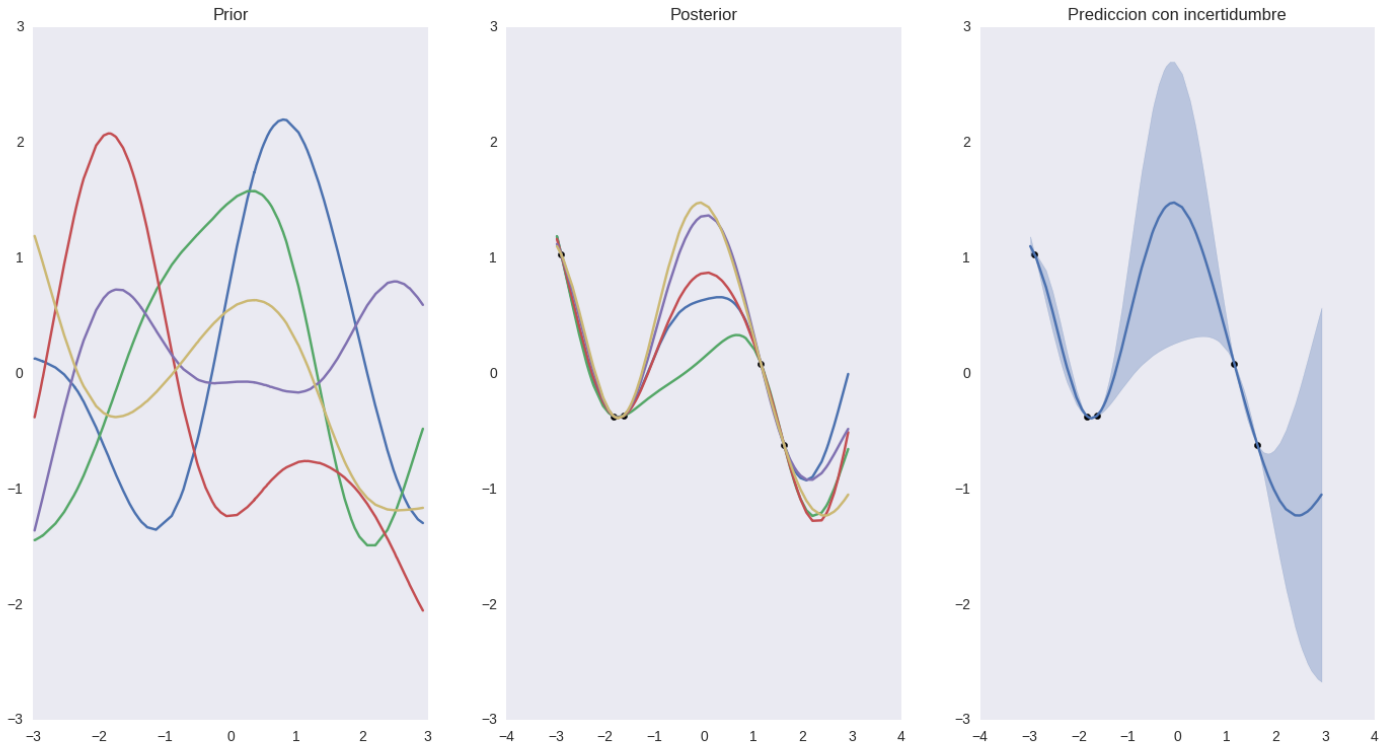


Figura 2.5: Ilustración del procedimiento de predicción con procesos gaussianos.

En la figura de la izquierda aparecen cinco muestras de la distribución a priori, generadas como en la sección anterior. En la figura central aparecen cinco muestras de la distribución posterior, despues de observar cinco pares (x_i, y_i). Por último, en la figura de la izquierda aparece una muestra tomada también de la misma distribución posterior, pero en este caso mostramos también la incertidumbre (en azul sombreado se representa el intervalo de confianza del 95 % para la muestra que aparece en amarillo en la figura central).

Por simplicidad, decimos $\boldsymbol{k}_* = \boldsymbol{k}(\boldsymbol{x}_*)$.

La predicción f_* se conoce a veces como *predicción lineal*, ya que si escribimos $\boldsymbol{\alpha} = (K + \sigma_n^2 I)^{-1}\boldsymbol{y}$, podemos escribir $\bar{f}_*$ como una combinación lineal de los $k(\boldsymbol{x}_i, \boldsymbol{x}_*)$:

$$\bar{f}(\boldsymbol{x}_*) = \sum_{i=1}^{n} \alpha_i k(\boldsymbol{x}_i, \boldsymbol{x}_*) \tag{2.13}$$

Para encontrar el valor de los hiperparámetros que mejor se ajustan a los datos, debemos optimizar la función de *verosimilitud marginal*, que es la siguiente:

$$p(\boldsymbol{y}|X) = \int p(\boldsymbol{y}|\boldsymbol{f}, X)p(\boldsymbol{f}|X)d\boldsymbol{f} \tag{2.14}$$

Este método (elegir los hiperparámetros que optimizan la función de verosimilitud marginal) se conoce como *selección bayesiana de modelos* e intentamos explicarlo a continuación, siguiendo [4, Capítulo 3]:

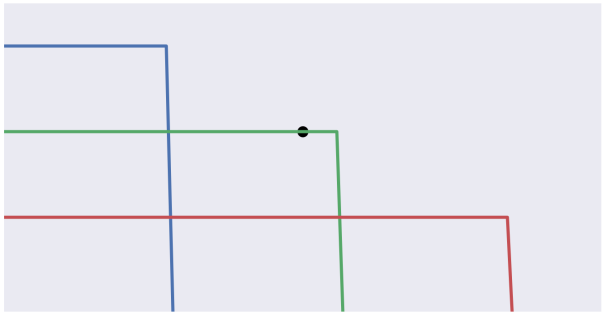


Figura 2.6: Ilustración del concepto de comparación bayesiana de modelos.

En la figura anterior, en el eje horizontal se representan todos los posibles datos, y en el vertical la verosimilitud marginal de un modelo para dichos datos. Se representan tres posibles modelos: azul (mayor complejidad), verde (complejidad intermedia) y rojo (menor complejidad).

Dado un conjunto de datos $\mathcal{D} = (X, \boldsymbol{y})$ y un conjunto de hiperparámetros Θ buscamos un modelo $\boldsymbol{f}$ que los explique los datos. Pudiendo calcular, para cada modelo posible $\boldsymbol{f}$, su verosimilitud $p(\boldsymbol{y}|\boldsymbol{f}, X)$ y su distibución a priori $p(\boldsymbol{f}|X)$, podemos elegir el modelo que optimiza la distribución posterior $p(\boldsymbol{f}|X, \boldsymbol{y}) \propto p(\boldsymbol{y}|\boldsymbol{f}, X)p(\boldsymbol{f}|X)$ y quedarnos con $\boldsymbol{f}_{MAP} = \arg\max p(\boldsymbol{f}|X, \boldsymbol{y})$. Este sería el modelo que mejor explica los datos. Sin embargo, precisamente por eso podríamos encontrarnos con un problema de *sobre-ajuste* y preferimos hacer una media ponderada de todos los modelos posibles, como en (2.14).

Optimizar la función de verosimlitud marginal es equivalente a optimizar su logaritmo, lo cuál es mucho más fácil porque sus derivadas son más fáciles de calcular. Utiliando que $\boldsymbol{y}|\boldsymbol{f}, X \sim \mathcal{N}(\boldsymbol{y}|\boldsymbol{f}, \sigma_f^2 I)$, $\boldsymbol{f}|X \sim \mathcal{N}(\boldsymbol{f}|\boldsymbol{0}, K)$, aplicamos la identidad (1.11) y obtenemos:

$$\log p(\boldsymbol{y}|X) = -\frac{1}{2}\boldsymbol{y}^T(K + \sigma_f^2 I)^{-1}\boldsymbol{y} - \frac{1}{2}\log|K + \sigma_f^2 I| - \frac{n}{2}\log(2\pi) \tag{2.15}$$

Normalmente, en las aplicaciones, buscamos predecir el valor de f en varios datos de test, $\boldsymbol{x}_{*1}, \boldsymbol{x}_{*2}, ..., \boldsymbol{x}_{*n_*}$. De esta forma, buscamos una expresión analítica para la distribución conjunta de $(\boldsymbol{f}, \boldsymbol{f}_*)^T$, siendo $\boldsymbol{f}_*$ un vector de tamaño n_*. Para ello definimos $K(X, X) = K$ y además:

$$K(X, X_*) = \begin{pmatrix} k(\boldsymbol{x}_1, \boldsymbol{x}_{*1}) & k(\boldsymbol{x}_1, \boldsymbol{x}_{*2}) & ... & k(\boldsymbol{x}_1, \boldsymbol{x}_{*n_*}) \\ k(\boldsymbol{x}_2, \boldsymbol{x}_{*1}) & k(\boldsymbol{x}_2, \boldsymbol{x}_{*2}) & ... & k(\boldsymbol{x}_2, \boldsymbol{x}_{*n_*}) \\ ... & ... & ... & ... \\ k(\boldsymbol{x}_n, \boldsymbol{x}_{*1}) & k(\boldsymbol{x}_n, \boldsymbol{x}_{*2}) & ... & k(\boldsymbol{x}_n, \boldsymbol{x}_{*n_*}) \end{pmatrix} \tag{2.16}$$

Definimos $K(X_*, X_*)$ de manera similar, y ya podemos obtener una expresión analítica para nuestra distribución conjunta:

$$\begin{pmatrix} \boldsymbol{f} \\ \boldsymbol{f}_* \end{pmatrix} \sim \mathcal{N}\left(\boldsymbol{0}, \begin{pmatrix} K(X, X) + \sigma^2 I & K(X, X_*) \\ K(X_*, X) & K(X_*, X_*) \end{pmatrix}\right) \tag{2.17}$$

En este punto recordamos las fórmulas (1.5)-(1.7) explicadas en la sección (1.3.2) y llegamos a una expresión analítica para la predicción $\boldsymbol{f}_*$:

$$\boldsymbol{f}_* \sim \mathcal{N}(\overline{\boldsymbol{f}_*}, cov(\boldsymbol{f}_*)) \tag{2.18}$$

$$\overline{\boldsymbol{f}_*} = K(X_*, X)[K(X, X) + \sigma^2 I]^{-1}\boldsymbol{y} \tag{2.19}$$

$$cov(\boldsymbol{f}_*) = K(X_*, X)[K(X, X) + \sigma^2 I]^{-1}K(X, X_*) \tag{2.20}$$

2.5. Clasificación con Procesos gaussianos

Un problema de clasificación consiste en asignar a un vector $\boldsymbol{x}$ una clase C_i de entre K posibles clases $C_1, C_2, ..., C_K$. En este trabajo nos centramos en problemas de **clasificación binaria**, en los que el objetivo es asignar a un vector $\boldsymbol{x}$ una clase de entre dos posibles, lo que representaremos con una etiqueta $y(\boldsymbol{x})$, que toma valores en $\{-1, 1\}$.

Sean $\{(\boldsymbol{x}_i, y_i)|i = 1, ..., n\}$ un conjunto de entrenamiento, con $y_i = y(x_i) \in \{-1, 1\}$ y $\boldsymbol{x}_*$ un dato de prueba. El objetivo es clasificar correctamente dicho dato. Para ello buscamos estimar la función $\pi(\boldsymbol{x}_*) = p(y_* = 1|\boldsymbol{x}_*)$.

Recordemos que un proceso gaussiano $f(\boldsymbol{x}) \sim \mathcal{GP}(m(\boldsymbol{x}), k(\boldsymbol{x}, \boldsymbol{x'}))$ define una función estocástica f que toma valores en el conjunto de los números reales. Sin embargo las probabilidades toman valores en el intervalo $[0, 1]$ y por tanto necesitamos una función $\sigma : (-\infty, \infty) \longrightarrow [0, 1]$ que sea *suave* y *estrictamente creciente*. Dicha función será la función logística, que se define de la siguiente manera:

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{2.21}$$

De esta manera, podemos escribir $\pi(\boldsymbol{x}_*) = p(y_* = 1|\boldsymbol{x}_*) = \sigma(f(\boldsymbol{x}_*))$.

Si queremos utilizar la metodología de *selección bayesiana de modelos*, explicada en (2.4), el proceso de inferencia se divide en dos pasos: primero estimamos la distribución posterior del proceso gaussiano, $p(f_*|X, \boldsymbol{y}, \boldsymbol{x}_*)$ y después la probabilidad en la que estamos interesados, $\pi(\boldsymbol{x}_*) = p(y_* = 1|X, \boldsymbol{y}, \boldsymbol{x}_*)$:

$$p(f_*|X, \boldsymbol{y}, \boldsymbol{x}_*) = \int p(f_*|X, \boldsymbol{x}_*, \boldsymbol{f})p(\boldsymbol{f}|X, \boldsymbol{y})d\boldsymbol{f} \tag{2.22}$$

$$\bar{\pi}_* = p(y_* = 1|X, \boldsymbol{y}, \boldsymbol{x}_*) = \int \sigma(f_*)p(f_*|X, \boldsymbol{y}, \boldsymbol{x}_*)df_* \tag{2.23}$$

En la primera de las dos integrales aparece un problema: la distribución posterior $p(\boldsymbol{f}|X, \boldsymbol{y})$ no es normal, y por tanto no podemos encontrar una forma analítica para la distribución y tendremos que aproximar $p(\boldsymbol{f}|X, \boldsymbol{y})$ con una distribución normal de forma que podamos evaluar las integrales anteriores de forma aproximada. Un método sencillo para hacer esto es la *aproximación de Laplace*, que discutimos en la siguiente sección.

2.6. Aproximación de Laplace

Hemos visto en la sección anterior que la distribución posterior $p(\boldsymbol{f}|X, \boldsymbol{y})$ no es gaussiana y por tanto no tenemos una fórmula cerrada que nos permita evaluar de forma analítica las integrales presentadas en la sección anterior. En esta sección presentamos un método sencillo para

aproximar la distribución posterior por una gaussiana, lo que nos permitirá calcular de forma aproximada $p(f_*|X, \boldsymbol{y}, \boldsymbol{x}_*)$. Por otro lado, la integral que define $p(y_* = 1|X, \boldsymbol{y}, \boldsymbol{x}_*)$ tampoco se puede evaluar (en general) de forma analítica y tendremos que utilizar métodos de cuadratura numérica para evaluar dicha integral.

El objetivo del método conocido como **aproximación de Laplace** es aproximar una distribución no-gaussiana $p(\boldsymbol{f}|X, \boldsymbol{y})$ por una distribución gaussiana $q(\boldsymbol{f}|X, \boldsymbol{y})$. Para ello hacemos una aproximación de Taylor de orden 2 de $\log p(\boldsymbol{f}|X, \boldsymbol{y})$:

$$q(\boldsymbol{f}|X, \boldsymbol{y}) = \mathcal{N}(\boldsymbol{f}; \hat{\boldsymbol{f}}, A^{-1}) \tag{2.24}$$

Siendo $\hat{\boldsymbol{f}} = \arg\max_{\boldsymbol{f}} q(\boldsymbol{f}|X, \boldsymbol{y})$ y $A = -\nabla\nabla \log p(\boldsymbol{f}|X, \boldsymbol{y})|_{\boldsymbol{f}=\hat{\boldsymbol{f}}}$.

Antes de ver como podemos utilizar la aproximación de Laplace para problemas de clasificación con procesos gaussianos, mostramos el concepto general con una figura:

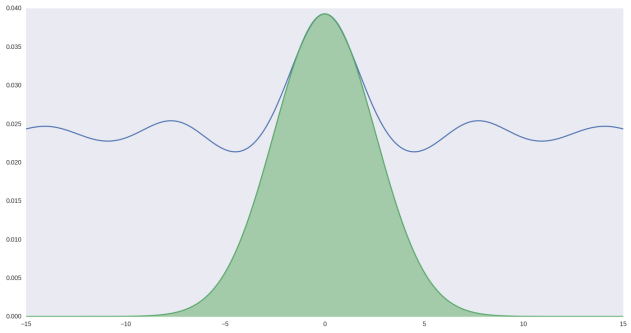


Figura 2.7: Ilustración de aproximación de Laplace.

En azul se muestra una distribución proporcional a $\exp(\frac{1}{2}\frac{sen(x)}{x})$, mientras que en verde se muestra su aproximación de Laplace. En la figura observamos que la aproximación tiene un problema: la distribución original tiene una cola mucho más pesada que la normal que la aproxima. Además, hay que tener en cuenta que la aproximación de Laplace será siempre simétrica, aunque la distribución que pretendemos aproximar no lo sea. Más detalles sobre la aproximación de Laplace pueden encontrarse en [5, Capítulo 27]

Aplicando el teorema de Bayes, $p(\boldsymbol{f}|X, \boldsymbol{y}) = p(\boldsymbol{y}|\boldsymbol{f})p(\boldsymbol{f}|X)/p(\boldsymbol{y}|X)$. Como el denominador es una constante y la función logaritmo es estrictamente creciente maximizar $\log p(\boldsymbol{f}|X, \boldsymbol{y})$ es equivalente a maximizar $\Psi(\boldsymbol{f})$, con $\Psi(\boldsymbol{f})$ definida de la siguiente manera:

$$\Psi(\boldsymbol{f}) = \log p(\boldsymbol{y}|\boldsymbol{f}) + \log p(\boldsymbol{f}|X) \tag{2.25}$$

. Para optimizar esta función, vamos a utilizar un método de quasi-Newton.

$$\nabla\Psi(\boldsymbol{f}) = \nabla \log p(\boldsymbol{f}|X, \boldsymbol{y}) - K^{-1}\boldsymbol{f} \tag{2.26}$$

$$\nabla\nabla\Psi(\boldsymbol{f}) = \nabla\nabla \log p(\boldsymbol{f}|X, \boldsymbol{y}) - K^{-1} = -W - K^{-1} \tag{2.27}$$

De esta forma podemos utilizar el método de Newton con la siguiente iteración:

$$\begin{aligned} \boldsymbol{f}^{(k+1)} &= \boldsymbol{f}^{(k)} + (K^{-1} + W)^{-1}(\nabla \log p(\boldsymbol{y}|\boldsymbol{f}^{(k)}) - K^{-1}\boldsymbol{f}^{(k)} \\ &= (K^{-1} + W)^{-1}(W\boldsymbol{f}^{(k)} + \nabla \log p(\boldsymbol{y}|\boldsymbol{f}^{(k)}) \end{aligned} \tag{2.28}$$

Ahora ya podemos definir explícitamente la aproximación de Laplace a $p(\boldsymbol{f}|X, \boldsymbol{y})$, $q(\boldsymbol{f}|X, \boldsymbol{y})$:

$$q(\boldsymbol{f}|X, \boldsymbol{y}) = \mathcal{N}(\boldsymbol{f}; \hat{\boldsymbol{f}}, (K^{-1} + W)^{-1}) \tag{2.29}$$

Siendo $K^{-1} + W = -\nabla\nabla \log p(\boldsymbol{f}|X, \boldsymbol{y})$ y $\hat{\boldsymbol{f}}$ el vector al que converge la aproximación de Newton anterior.

Ahora que tenemos la distribución posterior $q(\boldsymbol{f}|X, \boldsymbol{y})$ para los datos de entrenamiento, podemos calcular la distribución posterior $q(f_*|X, \boldsymbol{y}, \boldsymbol{x}_*)$ para un dato de prueba $\boldsymbol{x}_*$:

$$E[f_*|X, \boldsymbol{y}, \boldsymbol{x}_*] = \boldsymbol{k}(\boldsymbol{x}_*)^T K^{-1} \hat{\boldsymbol{f}} = \boldsymbol{k}(\boldsymbol{x}_*)^T \nabla \log p(\boldsymbol{y}|\hat{\boldsymbol{f}}) \tag{2.30}$$

$$V_q[f_*|X, \boldsymbol{y}, \boldsymbol{x}_*] = k(\boldsymbol{x}_*, \boldsymbol{x}_*) - \boldsymbol{k}(\boldsymbol{x}_*)^T (K + W^{-1})^{-1} \boldsymbol{k}(\boldsymbol{x}_*) \tag{2.31}$$

A la hora de implementar los algoritmos, es importante minimizar el tiempo de computación así como evitar cálculos inestables. Por ejemplo, debemos tener cuidado con el hecho de que la matriz K puede tener autovalores arbitrariamente pequeños. Además, invertir la matriz K con el algoritmo *clásico* tiene un coste $\mathcal{O}(n^3)$, siendo n el tamaño del conjunto de entrenamiento, lo que podría llevar más tiempo del que estamos dispuestos a esperar.

Dado que la matriz K es simétrica y definida positiva, podemos invertirla utilizando el método conocido como *descomposición de Cholesky*. De esta forma, podemos escribir K como producto de dos matrices triangulares: $K = LL^T$, siendo L una matriz triangular inferior que se conoce como *factor de Cholesky*. Esta descomposición nos permite además calcular el determinante de K de forma eficiente:

$$|K| = \prod_{i=1}^{n} L_{ii}^2, \text{ o también } \log|K| = 2\sum_{i=1}^{n} \log L_{ii} \tag{2.32}$$

Con el objetivo de obtener una implementación eficiente, definimos la siguiente matriz:

$$B = I + W^{\frac{1}{2}} K W^{\frac{1}{2}} \tag{2.33}$$

La motivación para calcular la matrix B es la siguiente: para calcular la matriz de covarianzas de la aproximación de Laplace $q(\boldsymbol{f}|X, \boldsymbol{y})$ tenemos que invertir la matriz K. La matriz K es semidefinida positiva y sus autovalores son todos números reales positivos. Sin embargo, sus autovalores pueden ser arbitrariamente pequeños y eso puede generar inestabilidades a la hora de calcular su matriz inversa o su determinante. Una solución para este problema consiste en añadir un *jitter* a la matriz de covarianza, como explicamos al comienzo del capítulo. Sin embargo, nosotros implementaremos una solución mejor tomada de [1, Capítulo 3] y para ello utilizaremos la matriz B definida anteriormente.. Como mostramos a continuación, no tendremos necesidad de invertir la matriz K y solamente tendremos que invertir la matriz B, cuyos autovalores están acotados inferiormente por 1 y superiormente por $1 + \frac{n}{4} max|K_{ij}|$. Para comprobar estas cotas tenemos que demostrar primero que cualquier vector z verifica la identidad $(1 + \frac{n}{4} max|K_{ij}|)||\boldsymbol{z}||^2 \geq \boldsymbol{z}^T B \boldsymbol{z} \geq ||\boldsymbol{z}||^2$ y después sustituir en dicha expresión $\boldsymbol{z}$ por los autovectores de B, obteniendo:

$$(1 + \frac{n}{4} max|K_{ij}|)||\boldsymbol{z}_j||^2 \geq \lambda_j ||\boldsymbol{z}_j||^2 \geq ||\boldsymbol{z}_j||^2 \tag{2.34}$$

Para calcular de forma eficiente la matriz de covarianzas de la aproximación de Laplace, $(K^{-1} + W)^{-1}$, utilizamos el lema de inversión de matrices (1.3):

$$(K^{-1} + W)^{-1} = K - K W^{\frac{1}{2}} B^{-1} W^{\frac{1}{2}} K \tag{2.35}$$

Para calcular de forma eficiente la inversa de B, calculamos su descomposición de Cholesky, $B = LL^T$ y podemos obtener su inversa como $B^{-1} = (L^{-1})^T L^{-1}$.

Ahora ya podemos calcular la variaza (bajo la aproximación de Laplace, q) de nuestras predicciones:

$$\mathbb{V}_q(f_*|\boldsymbol{y}) = k(\boldsymbol{x}_*, \boldsymbol{x}_*) - \boldsymbol{v}^T\boldsymbol{v}, \text{ con } v = L^{-1}W^{\frac{1}{2}}\boldsymbol{k}(\boldsymbol{x}_*) \tag{2.36}$$

Teniendo ya completamente especificada la distribución $q(f_*|X, y, \boldsymbol{x}_*)$ podemos obtener nuestra predicción $\bar{\pi}_*$, definida en (2.23), de forma aproximada:

$$\bar{\pi}_* \approx \int \sigma(f_*)q(f_*|X, y, \boldsymbol{x}_*)df_* = \int \sigma(z)\mathcal{N}(z; \bar{f}_*, V_q[f_*])dz \tag{2.37}$$

Evaluar esta integral analíticamente no es posible, sin embargo, podemos utilizar métodos de cuadratura para aproximarla con suficiente precisión.

2.7. Optimización de los hiperparámetros

Ya hemos visto como podemos afrontar un problema de clasificación utilizando procesos gaussianos fijando una función de covarianza. Nos queda por ver, por tanto, como elegir una función de covarianza "buena". La naturaleza del problema particular que estamos estudiando nos puede ayudar a escoger una función de covarianza u otra (por ejemplo, nos puede interesar elegir una función de covarianza que sea estacionaria, ...).

En la sección 2.2 hemos definido varias funciones de covarianza, que dependen de un conjunto de **hiperparámetros**, cuyos valores óptimos para una cierta tarea no suelen ser conocicos. Si queremos obtener el mejor resultado posible, debemos optimizar los hiperparámetros.

El enfoque que vamos a utilizar para optimizar los hiperparámetros de una cierta función de covarianza es el conocido como **selección bayesiana de modelos**, explicado en la sección 2.4:

Utilizamos una función de covarianza completamente especificada menos por sus hiperparámetros que quedarán "libres", y llamamos a esta hipótesis $\mathcal{H}$. Dado un conjunto de entrenamiento $\mathcal{D} = (X, \boldsymbol{y})$, buscamos los hiperparámetros Θ que maximicen la función de verosimilitud marginal $p(\boldsymbol{y}|X, \Theta, \mathcal{H})$.

Recordando lo visto en la sección 2.6, para un conjunto de hiperparámetros *fijo*, Θ, hemos aproximado la verosimilitud marginal, $p(\boldsymbol{y}|X)$ por la aproximación de Laplace $q(\boldsymbol{y}|X)$, y seguidamente hemos aproximado el logaritmo de la distribución posterior, $\log p(\boldsymbol{f}|X, \boldsymbol{y})$ por una función $\Psi(\boldsymbol{f})$. Escribiendo la aproximación de Taylor de orden 2 de $\Psi(\boldsymbol{f})$ alrededor de la moda $\hat{\boldsymbol{f}}$, obtenemos:

$$\Psi(\boldsymbol{f}) = \Psi(\hat{\boldsymbol{f}}) - \frac{1}{2}(\boldsymbol{f} - \hat{\boldsymbol{f}})^T A(\boldsymbol{f} - \hat{\boldsymbol{f}}) \tag{2.38}$$

Y de esta forma podemos obtener una aproximación $q(\boldsymbol{y}|X)$ a la verosimilitud marginal:

$$p(\boldsymbol{y}|X) \approx q(\boldsymbol{y}|X) = \exp(\Psi(\hat{\boldsymbol{f}})) \int \exp(-\frac{1}{2}(\boldsymbol{f} - \hat{\boldsymbol{f}})^T A(\boldsymbol{f} - \hat{\boldsymbol{f}}))d\boldsymbol{f} \tag{2.39}$$

Esta integral se puede evaluar analíticamente, obteniendo:

$$\log q(\boldsymbol{y}|X, \Theta) = \log p(\boldsymbol{y}|\hat{\boldsymbol{f}}) + \log p(\hat{\boldsymbol{f}}|X) + \frac{n}{2}\log(2\pi) - \frac{1}{2}\log|K^{-1} + W| \tag{2.40}$$

Utilizando que $|B| = |I_n + W^{\frac{1}{2}}KW^{\frac{1}{2}}| = |K||K^{-1} + W|$ obtenemos:

$$\log q(\boldsymbol{y}|X, \Theta) = -\frac{1}{2}\hat{\boldsymbol{f}}^T K^{-1}\hat{\boldsymbol{f}} + \log p(\boldsymbol{y}|\hat{\boldsymbol{f}}) - \frac{1}{2}\log|B| \tag{2.41}$$

Ahora, para elegir los parámetros óptimos, vamos a utilizar un método de optimización (por ejemplo, un método de quasi-Newton) para encontrar el conjunto de hiperparámetros Θ que maximiza el logaritmo de la función de verosimilitud marginal, $\log q(\boldsymbol{y}|X,\Theta)$, y para ello necesitamos el gradiente $\nabla \log q(\boldsymbol{y}|X,\Theta)$. Para ello, calculamos la derivada parcial de $\log q(\boldsymbol{y}|X,\Theta)$ respecto de cada uno de los hiperparámetros:

$$\frac{\partial \log q(\boldsymbol{y}|X,\Theta)}{\partial \theta_j} = \frac{\partial \log q(\boldsymbol{y}|X,\Theta)}{\partial \theta_j}\bigg|_{explcita} + \sum_{i=1}^{n} \frac{\partial \log q(\boldsymbol{y}|X,\Theta)}{\partial \hat{f}_i}\frac{\partial \hat{f}_i}{\partial \theta_j} \tag{2.42}$$

Utilizando las fórmulas para calcular la derivada de la matriz inversa y del determinante, obtenemos una fórmula analítica para las derivadas parciales que necesitamos. Para la primera expresión, podemos reescribir B como $B = W^{\frac{1}{2}}(W^{-1}+K)W^{\frac{1}{2}}$. Por tanto, $B^{-1} = W^{\frac{-1}{2}}(W^{-1}+K)^{-1}W^{\frac{-1}{2}}$ y $\frac{\partial B}{\partial \theta_j} = W^{\frac{1}{2}}\frac{\partial K}{\partial \theta_j}W^{\frac{1}{2}}$. Observamos que $\log p(\boldsymbol{y}|\hat{\boldsymbol{f}})$ no depende de θ_j y recordando además que podemos reordenar el producto de matrices para calcular la traza, obtenemos:

$$\frac{\partial \log q(\boldsymbol{y}|X,\Theta)}{\partial \theta_j}\bigg|_{explcita} = \frac{1}{2}\hat{\boldsymbol{f}}^T K^{-1}\frac{\partial K}{\partial \theta_j}K^{-1}\hat{\boldsymbol{f}} - \frac{1}{2}\text{traza}((W^{-1}+K)^{-1}\frac{\partial K}{\partial \theta_j}) \tag{2.43}$$

Para el siguiente término tenemos que tener en cuenta que $\log|B| = \log|K| + \log|K^{-1}+W|$ y que K no depende de $\hat{\boldsymbol{f}}$. Por tanto $\frac{\partial \log|B|}{\partial \hat{f}_i} = \frac{\partial \log|K^{-1}+W|}{\partial \hat{f}_i}$ y obtenemos:

$$\begin{aligned}\frac{\partial \log q(\boldsymbol{y}|X,\Theta)}{\partial \hat{f}_i} &= -\frac{1}{2}\frac{\partial \log|K^{-1}+W|}{\partial \hat{f}_i} \\ &= -\frac{1}{2}\text{traza}((K^{-1}+W)^{-1}\frac{\partial W}{\partial \hat{f}_i}) \\ &= -\frac{1}{2}[(K^{-1}+W)^{-1}]_{ii}\frac{\partial^3}{\partial f_i^3}\log p(\boldsymbol{y}|\hat{\boldsymbol{f}})\end{aligned} \tag{2.44}$$

Nos queda evaluar el gradiente $\frac{\partial \hat{\boldsymbol{f}}}{\partial \theta_j}$. Para ello recordamos que $\hat{\boldsymbol{f}} = K\nabla \log p(\hat{\boldsymbol{f}}|X,\boldsymbol{y})$ utiliamos la regla de la cadena:

$$\frac{\partial \hat{\boldsymbol{f}}}{\partial \theta_j} = \frac{\partial K}{\partial \theta_j}\nabla \log p(\boldsymbol{y}|\hat{\boldsymbol{f}}) + K\frac{\partial \nabla \log(\boldsymbol{y}|\hat{\boldsymbol{f}})}{\partial \hat{\boldsymbol{f}}}\frac{\partial \hat{\boldsymbol{f}}}{\partial \theta_j} \tag{2.45}$$

Por tanto:

$$(I - K\frac{\partial \nabla \log(\boldsymbol{y}|\hat{\boldsymbol{f}})}{\partial \hat{\boldsymbol{f}}})\frac{\partial \hat{\boldsymbol{f}}}{\partial \theta_j} = \frac{\partial K}{\partial \theta_j}\nabla \log p(\boldsymbol{y}|\hat{\boldsymbol{f}}) \tag{2.46}$$

Ahora aplicamos (2.26) y (2.27) para ver que $\frac{\partial \nabla \log(\boldsymbol{y}|\hat{\boldsymbol{f}})}{\partial \hat{\boldsymbol{f}}} = -W$ y por tanto:

$$\frac{\partial \hat{\boldsymbol{f}}}{\partial \theta_j} = (I+KW)^{-1}\frac{\partial K}{\partial \theta_j}\nabla \log p(\boldsymbol{y}|\hat{\boldsymbol{f}}) \tag{2.47}$$

2.8. Funciones de covarianza *Spectral Mixture Kernels*

Un problema que surge al utilizar procesos gaussianos es que el rendimiento del algoritmo depende de la función de covarianza elegida para aprender una determinada tarea. La función de covarianza elegida para aprender una tarea determina la manera de medir la *similitud* entre diferentes datos de entrada (ya sean estos de entrenamiento o de prueba).

Para aliviar este problema podemos utilizar funciones de covarianza que tengan la propiedad de ser **muy expresivas**. Esto quiere decir que pueden utilizarse para aprender muchas tareas diferentes, ya que pueden aproximar diferentes funciones de covarianza con una precisión arbitraria. Las funciones de covarianza que vamos a utilizar, propuestas en [3] tienen esta propiedad, y se conocen como *Spectral Mixture Kernels*. La base teórica de estos kernels está en el *Teorema de Bochner*, que enunciamos a continuación:

Teorema 2.6. *Sea k una función de variable compleja definida en R^P. Entonces k es una función de covarianza de un proceso estocástico continuo y débilmente estacionario si y solo si se puede escribir de la siguiente manera:*

$$k(\tau) = \int_{R^P} \exp(2\pi i s^T \tau)\psi(ds) \tag{2.48}$$

siendo ψ una medida finita y positiva.

Asumiendo la hipótesis del teorema, asociada a la medida ψ encontramos una función S denominada *densidad espectral* de k. Decimos que S y k son *duales de Fourier*, y podemos calcularlos analíticamente de la siguiente manera:

$$k(\tau) = \int S(s)e^{2\pi i s^T \tau} ds \tag{2.49}$$

$$S(s) = \int k(\tau)e^{-2\pi i s^T \tau} d\tau \tag{2.50}$$

Usando una mixtura de distribuciones normales podemos obtener una amplia variedad de funciones de covarianza. De hecho, se puede demostrar que las mixturas de gaussianas forman un subconjunto *denso* en el conjunto de todas las funciones de distribución. Como consecuencia de esto, el dual de dicho subconjunto es denso en el conjunto de las funciones de covarianza estacionarias. Esto significa que podemos utilizar una funcion de covarianza *Spectral Mixture Kernel* para aproximar cualquier función de covarianza estacionaria con precisión arbitraria. Lógicamente, cuanta más precisión necesitemos la mixtura de gaussianas tendrá que tener más componentes y por tanto más hiperparámetros, lo que tiene un impacto importante en el rendimiento de los algoritmos que utilicemos.

Consideremos la siguiente mixtura de dos gaussianas en dimensión 1:

$$\phi(s; \mu, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\{-\frac{1}{2\sigma^2}(s-\mu)^2\} \tag{2.51}$$

$$S(s) = \frac{1}{2}[\phi(s) + \phi(-s)] \tag{2.52}$$

Sustituyendo en la fórmula de los duales de Fourier obtenemos:

$$k(\tau) = \exp(-2\pi^2\tau^2\sigma^2)\cos(2\pi\tau\mu) \tag{2.53}$$

Más generalmente, si ϕ es una mixtura de Q gaussianas en R^P, donde la q-ésima componente tiene media $\boldsymbol{\mu}_q = (\mu_q^{(1)}, ..., \mu_q^{(P)})$ y matriz de covarianzas $K = \text{diag}(v_q^{(1)}, ..., v_q^{(P)})$, entonces:

$$k(\tau) = \sum_{q=1}^{Q} w_q \cos(2\pi\tau^T\mu_q) \prod_{p=1}^{P} \exp(-2\pi^2\tau_p^2 v_q^{(p)}), \tag{2.54}$$

siendo τ el vector P-dimensional $\tau = \boldsymbol{x} - \boldsymbol{x}'$

Esta función de covarianza combina la ventaja de ser expresivo (puede aproximar muchas covarianzas conocidas) con la de tener una forma sencilla que permite una fácil interpretación: los pesos w_q indican el peso relativo de cada uno de los componentes de la mixtura. Los inversos de las medias, μ_q son los periodos de cada componente, mientras que los inversos de las desviaciones típicas, $1/\sqrt{v_q}$ determinan la rapidez con la que varía cada componente al modificar los vectores de entrada.

Debemos implementar en el código las derivadas de la función de covarianza calculándolas analíticamente. Recordemos la fórmula de dicha función, 2.54:

$$k(\tau) = \sum_{q=1}^{Q} w_q \cos(2\pi\tau^T \mu_q) \prod_{p=1}^{P} \exp(-2\pi^2 \tau_p^2 v_q^{(p)}) \tag{2.55}$$

$$= \sum_{q=1}^{Q} w_q \cos(2\pi\tau^T \mu_q) \exp(-2\pi^2 \sum_{p=1}^{P} \tau_p^2 v_q^{(p)}) \tag{2.56}$$

$$\frac{\partial k(\tau)}{\partial w_q} = \cos(2\pi\tau^T \mu_q) \exp(-2\pi^2 \sum_{p=1}^{P} \tau_p^2 v_q^{(p)}) \tag{2.57}$$

$$\frac{\partial k(\tau)}{\partial \mu_q^{(p)}} = -2\pi\tau_p \sum_{q=1}^{Q} w_q \sin(2\pi\tau^T \mu_q) \exp(-2\pi^2 \sum_{p=1}^{P} \tau_p^2 v_q^{(p)}) \tag{2.58}$$

$$\frac{\partial k(\tau)}{\partial v_q^{(p)}} = -2\pi^2\tau_p^2 \sum_{q=1}^{Q} w_q \cos(2\pi\tau^T \mu_q) \exp(-2\pi^2 \sum_{p=1}^{P} \tau_p^2 v_q^{(p)}) \tag{2.59}$$

3

Aprendizaje multi-tarea

3.1. Introducción

En el campo del aprendizaje supervisado, dados un *conjunto de datos de entrenamiento* $\mathcal{D} = (X, \boldsymbol{y})$, un *dato de test* $\boldsymbol{x}_*$ y un algoritmo de aprendizaje $\mathcal{A}$, el objetivo es predecir un valor para $f_* = f(\boldsymbol{x}_*)$. El algoritmo que utilizamos para realizar esta predicción utiliza el conjunto de entrenamiento y además, una serie de suposiciones que hacen que, dado un mismo conjunto de entrenamiento, obtengamos resultados diferentes según el algoritmo de regresión o clasificación que utilicemos.

Por ejemplo, el conocido método de *regresión lineal* utiliza el supuesto de que $f(\boldsymbol{x})$ *es una función lineal de* $\boldsymbol{x}$. El método de *regresión con Procesos Gaussianos* utiliza el supuesto de que para n puntos cualesquiera $\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_n$ el vector aleatorio $\boldsymbol{f} = (f(\boldsymbol{x}_1), f(\boldsymbol{x}_2), ..., f(\boldsymbol{x}_n))^T$ sigue una distribución normal n-dimensional $\mathcal{N}_n(\boldsymbol{\mu}, \boldsymbol{\Sigma})$.

El conjunto de suposiciones que un cierto algoritmo de aprendizaje supervisado utiliza para predecir el valor de $y(\boldsymbol{x})$ se conoce en la literatura académica como *sesgo inductivo*. Es importante señalar que todos los algoritmos utilizan sesgo inductivo para realizar una predicción, y este sesgo es necesario ya que el valor de una función f puede ser cualquiera para un *input* $\boldsymbol{x}_*$ que no hemos visto previamente.

El enfoque tradicional en el campo del aprendizaje supervisado es unilizar un *conjunto de entrenamiento* $\mathcal{D} = (X, \boldsymbol{y})$, dónde $\boldsymbol{y}$ es un vector columna que contiene observaciones de una función desconocida f contaminadas (generalmente contaminadas por un cierto ruido) con el objetivo de *aprender* la función f que en esta sección llamaremos *tarea*.

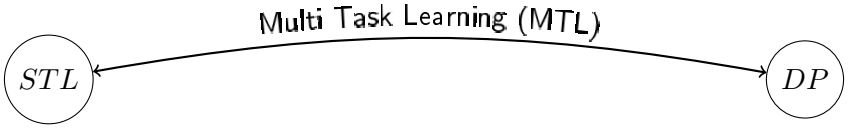
En este trabajo estudiamos un enfoque alternativo, que se conoce como *aprendizaje multi-tarea*. Un algoritmo de aprendizaje multi-tarea utiliza M conjuntos de entrenamiento $\mathcal{D}_m = (X_m, \boldsymbol{y}_m)(m = 1, 2, ..., M)$, para aprender al mismo tiempo M tareas diferentes $f_1, f_2, ..., f_M$. Con el objetivo de aprender las M tareas en paralelo, el algoritmo de aprendizaje multi-tarea utiliza una misma *representación* para todas ellas. En el caso particular de los procesos gaussianos, esta representación es un conjunto de hiperparámetros Θ.

Por tanto, para resolver con procesos gaussianos un problema unitarea (de regresión o de clasificación), tenemos que aplicar las técnicas aprendidas en el capítulo anterior para aprender los M conjuntos de hiperparámetros $\Theta_1, \Theta_2, ..., \Theta_M$ que optimizan las M funciones de vero-

similitud marginal $p(\boldsymbol{y}_m|X_m)$. Por el contrario, el enfoque multi-tarea consite en aprender el único conjunto de hiperparámetros Θ que optimiza una única función de verosimilitud marginal $p(\boldsymbol{y}|X)$.

El problema del aprendizaje multi-tarea ha sido estudiado previamente por varios autores y su utilidad ha sido comprobada, por ejemplo en [2].

Podríamos decir que la metodología del *Multi-Task Learning* es un enfoque intermedio entre el tradicional *Single Task Learning* y el *Data Pooling*. Supongamos que tenemos varios conjuntos de datos para aprender varias tareas que no están relacionadas entre sí: entonces el enfoque más adecuado será *STL*. Si tenemos varios conjutos de datos para una única tarea pero obtenidos de diferentes fuentes el enfoque más adecuado será *Data Pooling*: podemos juntar todos los datos y obtener un único clasificador. Con el aprendizaje multi-tarea no obtenemos un único clasificador sino un clasificador por tarea, con la diferencia de que estos utilizan una representación común para todas ellas y por ello también se aprende la similitud entre ellas.



3.2. Aprendizaje multi-tarea con procesos gaussianos

Para aprovechar las ventajas del *aprendizaje multi-tarea* en los algorimos de aprendizaje con procesos gaussianos asumimos lo siguiente: para aprender las tareas $y_1, y_2, ..., y_M$ utilizamos la misma función de covarianza con idénticos hiperparámetros. De esta forma, nuestro algoritmo aprenderá los hiperparámetros de la función de covarianza utilizando los M conjuntos de entrenamiento simultaneamente.

Si tuvieramos que aprender las M tareas individualmente, aplicaríamos los métodos descritos en el capítulo 2, definiendo para cada tarea un proceso gaussiano $\mathcal{GP}(0, k_m(\boldsymbol{x}, \boldsymbol{x}'))$ y tras un procedimiento de optimización del conjunto de hiperparámetros obtendríamos un vector aleatorio $\boldsymbol{f}$ distribuido según $\mathcal{N}(\mathbf{0}, K_m)$. Recordando que tenemos M conjuntos de entrenamiento $\mathcal{D}_m = (X_m, \boldsymbol{y}_m)$ asumimos que los patos para las distintas tareas han sido obtenidos de forma independiente y por tanto podemos escribir:

$$p(\boldsymbol{y}_1, ..., \boldsymbol{y}_M|X_1, ..., X_M) = \prod_{m=1}^{M} p(\boldsymbol{y}_m|X_m) \tag{3.1}$$

Por tanto, siguiendo [6], podemos definir un único proceso gaussiano de la siguiente forma: Escribimos un vector de observaciones $\boldsymbol{y} = [\boldsymbol{y}_1^T \dots \boldsymbol{y}_M^T]^T$ y una matriz de covarianza K de la siguiente forma:

$$K = \begin{bmatrix} K_1 & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{0} & K_1 & \dots & \mathbf{0} \\ \dots & \dots & \dots & \dots \\ \mathbf{0} & \mathbf{0} & \dots & K_M \end{bmatrix} \tag{3.2}$$

Ahora podemos entrenar un proceso gaussiano con las técnicas vistas en la sección 2.7 para entrenar los hiperparámetros que mejor se ajustan a los M conjuntos de datos. Una vez obtenemos los hiperparámetros óptimos, podemos utilizar las técnicas vistas en la sección 2.5 para

predecir la clase no conocida de un vector de datos para cualquiera de las tareas. En este sentido pensamos que será útil aprovechar la propiedad de *super-expresividad* de las funciones de covarianza *Spectral Mixture Kernel*, estudiadas en la sección 2.8.

Para entender mejor como funiona el entrenamiento de los M clasificadores al mismo tiempo, es buena idea representar los enfoques de *STL* y *MTL* en un *modelo probabilistico en grafos*, como hacemos a continuación, siguiendo [6]:

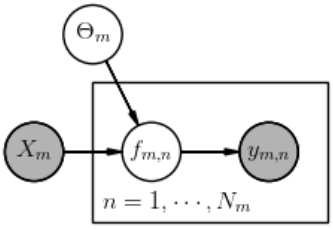


Figura 3.1: Representación gráfica de STL con procesos gaussianos.

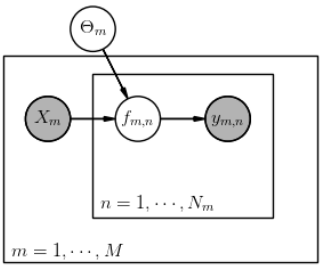


Figura 3.2: Representación gráfica de MTL con procesos gaussianos.

4

Sistema, diseño y desarrollo

4.1. El lenguaje Python, Numpy y Scipy

4.1.1. El lenguaje de programación Python

La implementación de los diferentes algoritmos que hemos presentado en este trabajo, así como el código necesario para realizar los experimentos que se detallan más adelante se han realizado utilizando **el lenguaje de programación Python**.

Python es un lenguaje de programación interpretado, de alto nivel y de propósito general. Soporta múltiples estilos de programación, incluyendo programación imperativa, procedural, orientada a objetos y funcional [7].

La implementación de referencia del lenguaje Python es **CPython**. Esta implementación es de código abierto y gratuita.

Python utiliza **tipado dinámico** (es decir, que la comprobación de tipificación se realiza en tiempo de ejecución y no durante la compilación). Otra de sus caracterísitcas más importantes es que dispone de un **recolector de basura**, y por tanto el programador no necesita preocuparse de liberar la memoria reservada.

Otra característica del lenguaje Python es que prioriza como objetivo la legibilidad del código y la facilidad de uso frente al rendimiento. Cuando el tiempo de ejecución es importante, el desarrollador tiene varias alternativas para implementar las funciones críticas:

1. Implementar las funciones críticas en lenguajes de programación como C.
2. Utilizar el compilador *Just in time* de Python, **PyPy**.
3. Utilizar el lenguaje **Cython**, un superconjunto de Python diseñado para escribir programas (principalmente) en Python obteniendo un rendimiento próximo al de C.

Uno de los puntos fuertes del lenguaje Python es que dispone de una librería estándar bastante grande. Sin embargo, para manejar cómodamente objetos matemáticos como vectores y matrices, así como para utilizar rutinas de cálculo numérico y optimización, es necesario utilizar otras biliotecas, como **Numpy** y **Scipy**, cuyo funcionamiento se detalla en la siguiente subsección.

4.1.2. Los paquetes Numpy y Scipy

El paquete **Numpy** es un paquete de Python para computación científica. Incluye objetos matemáticos como vectores, matrices o vectores multidimensionales, así como una colección de rutinas para realizar de manera eficiente operaciones con vectores y matrices, incluyendo operaciones básicas de álgebra lineal, estadística y simulación aleatoria.

El *núcleo* del paquete Numpy es es objeto *ndarray*. Este objeto encapsula vectores de tamaño arbitrario que contienen objetos del mismo tipo. Los objetos de tipo *ndarray* son diferentes de las diferentes *secuencias* de Python (incluyendo los objetos de tipo *list* o *tuple*), ya que los primeros están diseñados con el objetivo de realizar operaciones sobre ellos con el rendimiento como prioridad.

El paquete **Scipy** es una colección de algoritmos matemáticos construida sobre la base del paquete Numpy. Está dividido en subpaquetes útiles para diferentes áreas de la computación científica, incluyendo subpaquetes de álgebra lineal, de estadística y de optimización. Uno de los métodos que proporciona el paquete Scipy es **scipy.optimize.fmin_l_bfgs_b**. Esta es una implementación optmizada del algoritmo **L-BFGS**, del que hablamos en la sección 1.3.2.

Una de las ventajas de utilizar Scipy es que al compilar nuestros programas podemos elegir la opción de utilizar la biblioteca **OpenBLAS**. OpenBLAS es una biblioteca de código abierto que incluye rutinas de álgebra lineal muy optimizadas.

4.2. Implementación de Procesos Gaussianos en Python

Con el objetivo de realizar experimentos que nos ayuden a confirmar la utilidad del aprendizaje multi-tarea en el contexto del aprendizaje supervisado con procesos gaussianos, hemos implementado un módulo de Python, **gp.py**, que pasamos a describir a contuación:

- Una clase **CovFunction**, con métodos **covarianceMatrix** y **pderMatrix**, que reciben una matriz de diseño X y devuelven las matrices K y $\partial K/\partial\theta_j$, respectivamente.
- La clase **SMKernel** (por *Spectral Mixture Kernel*) es una subclase de **CovFunction**. Se implementan las fórmulas de la función de covarianza y de sus derivadas parciales en los métodos **covFunction** y **computePder**, respectivamente. Estos métodos reciben como argumento dos vectores de entrada, así como los hiperparámetros, ya que representan funciones paramétricas.
- La clase **SigmoidFunction** es una superclase para las diferentes funciones sigmoidales (funciones $\sigma : R \longrightarrow [0,1]$, como la función logística o la función de distribución de la normal $\mathcal{N}(0,1)$) que se pueden utilizar para resolver problemas de clasificación.
- La clase **LogisticFunction** es una subclase de **SigmoidFunction**. Se implementan métodos para evaluar la función logística en un punto y para calcular el logaritmo de la función de verosimilitud $\log p(\boldsymbol{y}|X,\theta)$, así como su gradiente y hessiano, necesarios para los algoritmos que se describen en la siguiente subsección.
- La clase **GaussianProcess** lleva asociada una función de covarianza, que se le asigna en el constructor, y unas tareas, que se añaden con el método **addTask**. Cada tarea consiste en un para $(X,\boldsymbol{y})$, como se ha descrito en el capítulo anterior. En esta clase están implementados los algoritmos que se detallan en la siguiente sección.

Ahora que ya tenemos un módulo con las clases y métodos definidos, podemos pasar a implementar los algortimos, y a realizar los experimentos.

4.3. Implementación del algoritmo de clasificación

Recordando el estudio previo que hemos hecho del método de clasificación con GPs en la sección 2.5, el primer paso para resolver un problema de clasificación consiste en encontrar la moda $\hat{\boldsymbol{f}}$ de la distribución $p(\boldsymbol{f}|X,\boldsymbol{y})$ con el objetivo de aproximarla por una distribución normal $q(\boldsymbol{f}|X,\boldsymbol{y})$ utilizando el método de la aproximación de Laplace. Para ello utilizamos el siguiente algoritmo, extraído de [1, Capítulo 3], y adaptado para el aprendizaje multitarea:

Algorithm 1 Búsqueda de la moda para la aproximación de Laplace

procedure GPC-FIND-MODE
  $\log q(\boldsymbol{y}|X,\theta) \leftarrow 0$
  $\boldsymbol{f} \leftarrow \boldsymbol{0}$
  for each task **do**
    repeat
      $W \leftarrow -\nabla\nabla \log p(\boldsymbol{y}|\boldsymbol{f})$
      $L \leftarrow \text{cholesky}(I + W^{\frac{1}{2}} K W^{\frac{1}{2}})$
      $\boldsymbol{b} \leftarrow W\boldsymbol{f} + \nabla \log p(\boldsymbol{y}|\boldsymbol{f})$
      $\boldsymbol{a} \leftarrow \boldsymbol{b} - W^{\frac{1}{2}}(L^T)^{-1}L^{-1}(W^{\frac{1}{2}}K\boldsymbol{b})$
      $\boldsymbol{f} \leftarrow K\boldsymbol{a}$
    until convergence
    $\log q(\boldsymbol{y}|X,\theta) \leftarrow \log q(\boldsymbol{y}|X,\theta) - \frac{1}{2}\boldsymbol{a}^T\boldsymbol{f} + \log p(\boldsymbol{y}|\boldsymbol{f}) - \sum \log L_{ii}$
  end for
  $\hat{\boldsymbol{f}} \leftarrow \boldsymbol{f}$
  return $\hat{\boldsymbol{f}}, \boldsymbol{a}$

Una vez que hemos encontrado la moda, $\hat{\boldsymbol{f}}$, de la aproximación de Laplace, $q(\boldsymbol{y}|X,\theta)$, es utilizamos el método estudiado en la sección 2.5 para encontrar la probabilidad $\pi_* = (y_* = 1|\boldsymbol{x}_*,\theta)$. Para ello utilizamos el siguiente algoritmo, obtenido de nuevo de [1, Capítulo 3], pero adaptado para el aprendizaje multi-tarea:

Algorithm 2 Predicción para clasificación binaria con GPC

procedure GPC-MAKE-PREDICTION
  $\hat{\boldsymbol{f}} \leftarrow$ **GPC-FIND-MODE**
  $W \leftarrow -\nabla\nabla \log p(\boldsymbol{y}|\hat{\boldsymbol{f}})$
  $L \leftarrow \text{cholesky}(I + W^{\frac{1}{2}} K W^{\frac{1}{2}})$
  $\bar{f}_* \leftarrow \boldsymbol{k}(\boldsymbol{x}_*)^T \nabla \log p(\boldsymbol{y}|\hat{\boldsymbol{f}})$
  $\boldsymbol{v} = L^{-1}W^{\frac{1}{2}}\boldsymbol{k}(\boldsymbol{x}_*)$
  $\mathcal{V}(f_*) = k(\boldsymbol{x}_*,\boldsymbol{x}_*) - \boldsymbol{v}^T\boldsymbol{v}$
  $\bar{\pi}_* = \int \sigma(z)\mathcal{N}(z;\bar{f}_*,\mathcal{V}(f_*))dz$
  return $\bar{\pi}_*$

Una vez aprendidos los hiperparámetros, podemos utilizar el algoritmo 2 (que a su vez utiliza el algoritmo 1) para hacer predicciones. Sin embargo, como hemos estudiado, debemos utilizar los hiperparámetros que sean óptimos para las tareas que queremos aprender. Para ello, utilizamos el siguiente algoritmo, que calcula el logaritmo de la función de verosimilitud, $\log q(\boldsymbol{y}|X,\theta)$ y sus derivadas parciales. De esta forma podremos utilizar una rutina de optimización, como **L-BFGS**, que está implementada en el paquete SciPy, para encontrar los hiperparámetros θ óptimos para nuestras tareas. El siguiente algoritmo ha sido obtenido de[1, Capítulo 5], y al igual que los anteriores lo hemos adaptado para aprender múltiples tareas en paralelo:

Algorithm 3 Cálculo de $\log q(\boldsymbol{y}|X,\theta)$ y su gradiente

procedure GPC-COMPUTE-LOG-ML
  $\hat{\boldsymbol{f}} \leftarrow$ **GPC-FIND-MODE**
  $W \leftarrow -\nabla\nabla \log p(\boldsymbol{y}|\boldsymbol{f})$
  $L \leftarrow \text{cholesky}(I + W^{\frac{1}{2}} K W^{\frac{1}{2}})$
  $\log Z \leftarrow \frac{1}{2}\boldsymbol{a}^T \boldsymbol{f} + \log p(\boldsymbol{y}|\boldsymbol{f}) - \sum \log(\text{diag}(L))$
  $Z \leftarrow W^{\frac{1}{2}}(L^T)^{-1}(L^{-1}W^{\frac{1}{2}})$
  $C \leftarrow L^{-1}W^{\frac{1}{2}}K$
  $\boldsymbol{s_2} \leftarrow= -\frac{1}{2}\text{diag}(\text{diag}(K) - diag(C^T C))\nabla^3 \log p(\boldsymbol{y}|\boldsymbol{f})$
  for each j in 1, ..., dim($\boldsymbol{\theta}$) **do**
    $C \leftarrow \frac{\partial K}{\partial \theta_j}$
    $s_1 \leftarrow \frac{1}{2}\boldsymbol{a}^T C \boldsymbol{a} - \frac{1}{2}\text{traza}(RC)$
    $\boldsymbol{b} \leftarrow C\nabla \log p(\boldsymbol{y}|\boldsymbol{f})$
    $\boldsymbol{s}_3 \leftarrow \boldsymbol{b} - KR\boldsymbol{b}$
    $\nabla_j \log Z \leftarrow s_1 + \boldsymbol{s}_2^T \boldsymbol{s}_3$
  end for
  return $\log Z, \nabla \log Z$

5

Experimentos Realizados y Resultados

5.1. Experimentos con datos sintéticos

Con el objetivo de verificar que los métodos estudiados funcionan, antes de probarlos con datos reales hemos realizado experimentos con datos sintéticos que nos ayudan a verificar que han sido implementados correctamente.

Para ello utilizamos un problema generado utizando un proceso gaussiano con un kernel *Squared Exponential*:

$$k(\boldsymbol{x}, \boldsymbol{z}) = \sigma_f^2 \exp(-\frac{1}{2l^2}||\boldsymbol{x} - \boldsymbol{z}||^2)$$

Ahora generamos datos sintéticos de la siguiente manera:

Generamos 600 muestras de una distribución uniforme en el subconjunto $[-1, 1] \times [-1, 1]$ del plano real:

$$\boldsymbol{x}_1, ..., \boldsymbol{x}_{600} \sim \mathcal{U}([-1, 1] \times [-1, 1])$$

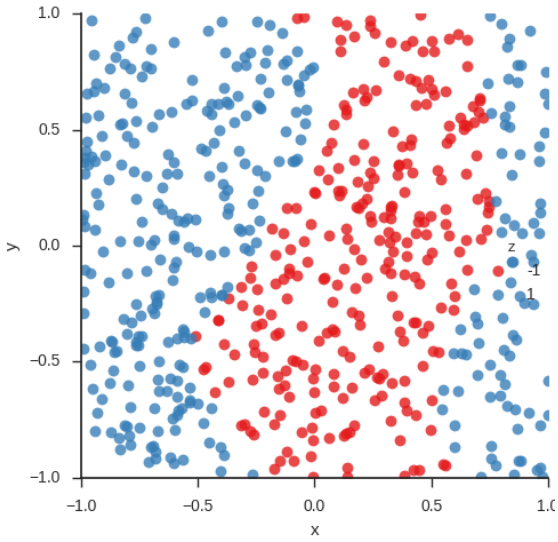
Ahora, para las 600 muestras, calculamos la matriz de Gram K y obtenemos una tarea de aprendizaje de la siguiente manera:

$$\boldsymbol{f} = (f_1, ..., f_{600})^T \sim \mathcal{N}(\boldsymbol{0}, K)$$

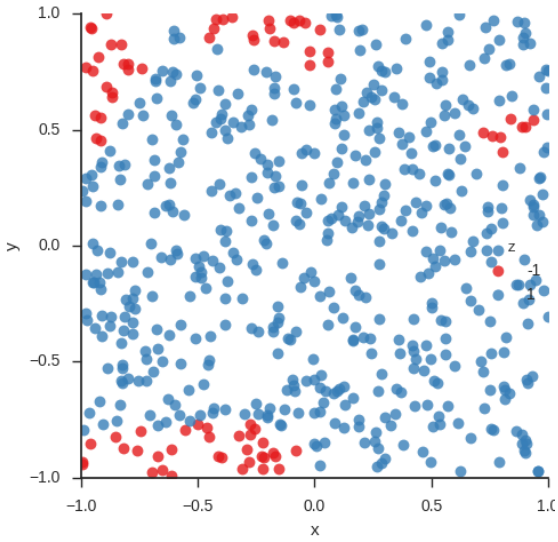
$$y_i = y(f_i) = \begin{cases} 1 & f_i \geq 0 \\ -1 & f_i < 0 \end{cases} \text{ para } i = 1, ..., 600 \tag{5.1}$$

Ahora ya tenemos un vector de observaciones $\boldsymbol{y} = (y_1, ..., y_{600})^T$. Con esto, dividimos nuestro conjunto de datos en dos subconjuntos de entrenamiento (con cien datos). y de pruebas (con 500 datos). Para el conjunto de entrenamiento además añadimos un cierto ruido logístico al vector $\boldsymbol{f}$, de forma que la tarea de aprendizaje sea más complicada.

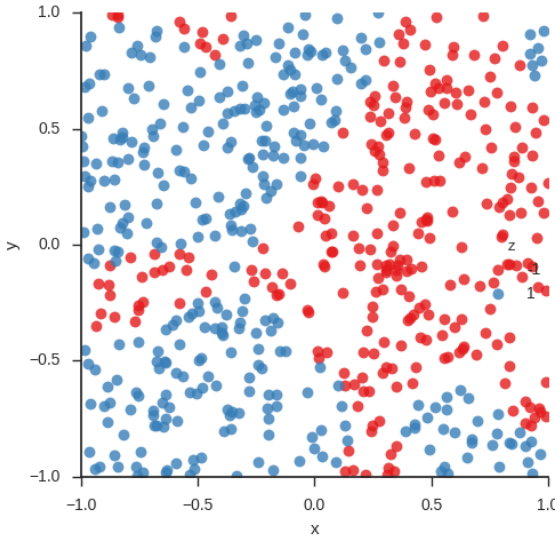
Este procedimientose repite cuatro veces, de forma que se generan cuatro tareas diferentes pero similares. Las cuatro tareas de aprendizaje se muestran en la siguiente página:



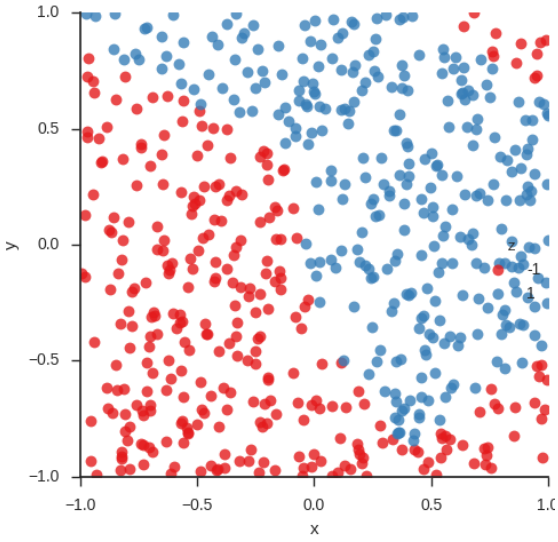
(a) Conjunto de datos 1



(b) Conjunto de datos 2



(c) Conjunto de datos 3



(d) Conjunto de datos 1

Figura 5.1: Cuatro tareas diferentes pero similares, generadas con datos sintéticos.

A continuación mostramos los resultados de aplicar aprendizaje uni-tarea (STL) a las cuatro tareas por separado y aprendizaje multi-tarea (MTL) de las cuatro tareas al mismo tiempo:

La siguiente tabla muestra el número de datos de pruebas clasificados correctamete (sobre un total de 500 datos):

	TAREA 1	TAREA 2	TAREA 3	TAREA 4
STL	339	432	339	259
MTL	370	437	348	444
MEJORA	**+31**	**+5**	**+9**	**+185**

Cuadro 5.1: Datos de prueba clasificados correctamente.

Para comparar mejor los resultados, mostramos en la siguiente tabla la tasa de aciertos:

	TAREA 1	TAREA 2	TAREA 3	TAREA 4
STL	67.8	86.4	67.8	51.8
MTL	74.0	87.4	69.6	88.8
MEJORA	**+6.2**	**+1.0**	**+1.8**	**+37.0**

Cuadro 5.2: Tasas de acierto para las diferentes tareas.

Observamos que en el aprendizaje de todas las tareas se experimenta una mejora cuando se aprenden todas a la vez. La mejora es especialmente significativa en las tareas más difíciles, especialmente en la última.

5.2. Clasificacion de datos de minas terrestres

Si en la sección anterior hemos comprobado que el aprendizaje multi-tarea puede mejorar el rendimiento (entendido como *tasa de acierto*) en tareas de clasificación, ahora pretendemos comprobar la utilidad del aprendizaje multi-tarea con un conjunto de datos real.

En [8] se presenta un conjunto de datos de **campos de minas terrestres**. Los datos han sido obtenidos de forma separada en 29 campos de minas terrestres diferentes. Las 29 tareas son de clasificación binaria y las observaciones (entre 447 y 690 para cada tarea) tienen 9 variables, todas ellas obtenidas a partir de imágenes de radar y las clases son 1 para **presencia de minas** y -1 para **ausencia de minas**. De las 29 tareas, las primeras 15 corresponden a terrenos rocosos, y las 14 últimas a terrenos desérticos. Esperamos que cada uno de estos dos grupos esté compuesto por tareas similares.

Cabe mencionar aquí que, dado que estas tareas son muy similares, podríamos plantearnos utilizar *data pooling* en lugar de *MTL*: podríamos juntar los datos de las quince tareas y obtener un único clasificador que nos sirva para todas ellas. Sin embargo, al utilizar *MTL* ahorraremos en tiempo de computación debido a que el coste de *data pooling* es cúbico, mientras que el coste de *MTL* es lineal: MTL para las 15 primeras tareas tendrá el mismo coste que STL para cada una de ellas (en ambos casos la operación más costosa es invertir 15 matrices de tamaño n), mientras que *data pooling* tendrá un coste $15^3 = 3375$ veces el coste de aprender una tarea, o lo que es lo mismo $15^2 = 225$ veces el coste de MTL.

Como primer experimento, utilizamos el algoritmo de procesos gaussianos para aprender las 15 tareas del primer grupo. Para cada tarea, utilizamos un conjunto de entrenamiento muy pequeño (con solamente 10 observaciones), y utilizamos el resto de observaciones como conjunto de

pruebas. Primero entrenamos un proceso gaussiano para cada tarea (STL) y después aplicamos el aprendizaje multi-tarea. Los resultados obtenidos son los siguientes:

	Tasa Acierto STL	Tasa Aciertos MTL	Mejora
TAREA 1	56.57	94.33	+37.76
TAREA 2	56.87	92.99	+36.12
TAREA 3	94.02	94.32	+0.30
TAREA 4	91.80	96.52	+4.72
TAREA 5	65.03	95.71	+30.68
TAREA 6	93.87	94.27	+0.40
TAREA 7	52.04	94.49	+42.45
TAREA 8	94.50	94.70	+0.20
TAREA 9	93.44	93.85	+0.41
TAREA 10	50.72	96.93	+46.21
TAREA 11	94.32	94.32	+0.00
TAREA 12	59.88	95.06	+35.18
TAREA 13	94.46	94.67	+0.21
TAREA 14	93.67	93.88	+0.21
TAREA 15	96.30	96.51	+0.21

Cuadro 5.3: Tareas de terrenos rocosos

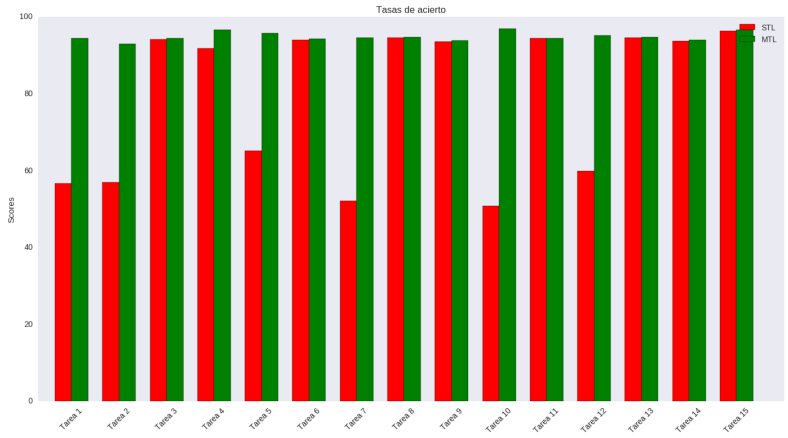


Figura 5.2: Mejora en la tasa de aciertos para las tareas 1-15. En rojo, tasa de acierto para STL. En verde, para MTL.

	Tasa Acierto STL	Tasa Aciertos MTL	Mejora
TAREA 16	93.41	93.41	0.00
TAREA 17	94.17	94.17	0.00
TAREA 18	94.39	93.46	0.07
TAREA 19	52.21	93.71	+41.50
TAREA 20	53.85	92.31	+38.46
TAREA 21	58.70	93.74	+35.04
TAREA 22	92.17	91.71	-0.46
TAREA 23	55.04	95.78	+40.74
TAREA 24	91.61	91.61	+0.00
TAREA 25	62.12	93.41	+31.29
TAREA 26	89.25	92.29	+3.04
TAREA 27	91.59	91.82	+0.23
TAREA 28	93.78	94.47	+0.69
TAREA 29	90.44	90.91	+0.47

Cuadro 5.4: Tareas de terrenos desérticos

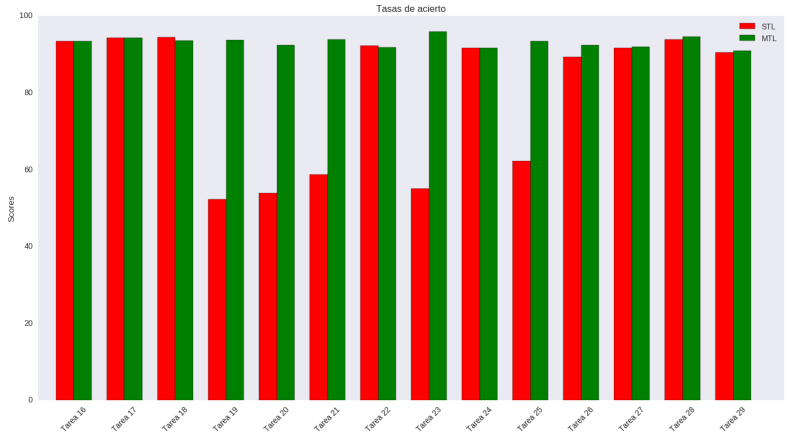


Figura 5.3: Mejora en la tasa de aciertos para las tareas 16-29. En rojo, tasa de acierto para STL. En verde, para MTL.

6

Conclusiones y trabajo futuro

Con este trabajo hemos comprobado que, en el contexto de los problemas de clasificación, el enfoque del aprendizaje multi-tarea ayuda a mejorar el rendimiento de los modelos conocidos como procesos gaussianos. Hemos comprobado también la utilidad de las funciones kernel *Spectral Mixture*, que nos han dado tasas de acierto superiores al 90 % en el problema de clasificación de datos de campos de minas terrestres aún utilizando conjuntos de datos muy pequeños.

En concreto, los experimentos realizados con datos de campos de minas indican que es posible obtener tasas de acierto muy grandes para varias tareas con conjuntos de datos muy pequños para cada una de ellas si todas las tareas se aprenden a la vez. Cabe recordar que el orden del algoritmo de clasificación mediante procesos gaussianos es $\mathcal{O}(n^3)$, siendo n el tamaño del conjunto de entrenamiento, así que no es realista pensar en utilizar procesos gaussianos con conjuntos de datos demasiado grandes. Por tanto, utilizando el enfoque multi-tarea podemos obtener tasas de acierto buenas sin necesidad de utilizar conjuntos de datos grandes, y por tanto sin necesidad de esperar tiempos de ejecución demasiado largos.

Como posible trabajo futuro sería interesante aplicar el enfoque multi-tarea a problemas de clasificación *multiclase* [1, Capítulo 3], así como utilizar métodos de inferencia aproximada diferentes de la aproximación de Laplace, siendo los más relevantes *Expectation Propagation* y los *métodos de Monte Carlo con cadenas de Markov* [1, Capítulo 3].

Bibliografía

[1] Christopher K. I. Rasmussen, Carl E.; Williams. *Gaussian Processes for Machine Learning*. The MIT Press, 2006.

[2] R. Caruana. Multitask learning. *Learning to learn*, pages 95–133, 1998.

[3] R. P. Wilson, A. G.; Adams. Gaussian process kernels for pattern discovery and extrapolation. *ICML*, pages 1067–1075, 2013.

[4] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[5] D. J. MacKay. *Information theory, inference and learning algorithms*. Cambridge University Press, 2003.

[6] Neil D.; John C. Platt Lawrence. Learning to learn with the informative vector machine. *Proceedings of the twenty-first international conference on Machine learning*, pages 35–63, 2004.

[7] G van Rossum. Python tutorial. *Centrum voor Wiskunde en Informatica*, 1995.

[8] Liao X. Carin L. Krishnapuram B. Xue, Y. Multi-task learning for classification with dirichlet process priors. *Journal of Machine Learning Research*, pages 35–63, 2007.